Synthesis Variables and Attributes

Version C-2009.06, June 2009

SYNOPSYS®

Table of Contents

MasterInstance

TYPE

string

DESCRIPTION

If a design has multiple instances, one way to resolve the multiple instances before compiling (when using Automated Chip Synthesis) is to select a master instance. Selecting a master instance is similar to using the **compile-dont_touch** method of resolving instances. The master instance is compiled and used for all of the instances referencing that design.

Use the **sub_designs_of** command to determine if your design contains multiple instances of any subdesign. Use the **sub_instances_of** command to identify the multiple instances.

Use the **set_attribute** command to set the **MasterInstance** attribute on the instance you want to use as the master instance.

You can also resolve multiple instances by uniquifying or ungrouping.

SEE ALSO

sub_designs_of (2), **sub_instances_of** (2), **ungroup** (2), **uniquify** (2).

access_internal_pins

Controls the creation, deletion, and user access of internal pins.

TYPE

Boolean

DEFAULT

false

GROUP

system_variables

DESCRIPTION

The **access_internal_pins** variable controls the creation and deletion of internal pins and the access of users to internal pins. Internal pins have to do with internal design inside a library cell. Because of the source of internal pins, they are created or deleted during link time, depending on the setting of this variable. Use the **find** command and **get_pins** command to show internal pins, if such pins exist. Certain timing commands can also set constraints on internal pins. When the **access_internal_pins** variable is set to *false* (the default), after the creation of internal pins, the tool deletes the pins immediately and the constraints on the pins are lost.

SEE ALSO

find(2)
get_pins(2)
link(2)

acs_area_report_suffix

Specifies the suffix for area reports generated during the automated compile process.

TYPE

string

DEFAULT

area

GROUP

acs_variables

DESCRIPTION

Specifies the suffix for area reports generated during the automated compile process. For use in dc_shell-t (Tcl mode of dc_shell) only. The default is *area*.

Unless you specify the **-no_reports** option, the compile scripts generated by the **write_compile_script** command include a command to generate an area report. The area report is placed in a file called pass*n*/reports/*design.suffix*, where *suffix* is the value of the **acs_area_report_suffix** variable.

To determine the current value of this variable, type **printvar acs_area_report_suffix**.

SEE ALSO

write_compile_script(2)

acs_autopart_max_area

Defines partition threshold; used with other acs variables to control chip-level partitioning.

TYPE

string

DEFAULT

0.0

GROUP

acs_variables

DESCRIPTION

Defines partition threshold; used with other acs variables to control chip-level partitioning. For use in **dc_shell-t** (Tcl mode of dc_shell) only.

To determine the current value of this variable, type **printvar acs_autopart_max_area**. If no maximum area is needed, set this variable to 0.0 (default).

SEE ALSO

read_partition(2)
report_partitions(2)
write_makefile(2)
write_partition(2)
write_partition_constraints(2)
acs_autopart_max_percent(3)

acs_autopart_max_percent

Controls chip-level partitioning; used with other acs variables.

TYPE

string

DEFAULT

0.0

GROUP

acs_variables

DESCRIPTION

Controls chip-level partitioning; used with other acs variables. For use in **dc_shell-t** (Tcl mode of dc_shell) only. The partition threshold is set as the percentage of total design area.

To determine the current value of this variable, type **printvar acs_autopart_max_percent**. If no maximum percentage specification is needed, set this variable to 0.0 (default).

SEE ALSO

read_partition(2)
write_makefile(2)
write_partition(2)
write_partition_constraints(2)
acs_autopart_max_area(3)

acs_budgeted_cstr_suffix

Specifies the suffix for constraint files generated by the **derive_partition_budgets** command.

TYPE

string

DEFAULT

con

GROUP

acs_variables

DESCRIPTION

Specifies the suffix for constraint files generated by the **derive_partition_budgets** command. For use in dc_shell-t (Tcl mode of dc_shell) only.

The **derive_partition_budgets** command generates a constraint file for each compile partition. The generated constraint files are named pass*n*/constraints/*design*.*suffix*, where *suffix* is the value of the **acs_budgeted_cstr_suffix** variable.

To determine the current value of this variable, type **printvar acs_budgeted_cstr_suffix**.

SEE ALSO

acs_compile_attributes

Customize the Automated Chip Synthesis compile strategy.

DESCRIPTION

Automated Chip Synthesis provides attributes that can customize the compile strategy. To set these attributes, use the **set_attribute** command to set the attribute on the reference design for the compile partition (or on the top-level design).

Each chip-level compile command (acs_compile_design, acs_refine_design, and acs_recompile_design) performs a two-step compile. The acs_compile_design and acs_recompile_design commands perform a full compile followed by a boundary compile. The acs_refine_design command performs an incremental compile followed by a boundary compile. The Automated Chip Synthesis compile attributes control both the effort and the strategy used in each of the steps.

These attributes affect the compile scripts generated by the **write_compile_script** command (and therefore the compile scripts generated by the chip-level compile commands). Some attributes add commands to the script before the compile command; others modify the compile command options.

Automated Chip Synthesis ignores these compile attributes if a custom compile strategy or custom compile script exists in the directory defined for the user_compile_strategy_script or user_override_script directories (default directory is $acs_work_dir/scripts/dest_dir).

The following attributes control the effort used in each compile step (full, incremental, and boundary):

• FullCompile
Valid values are none, low, medium, and high.
Default value is medium.

• IncrementalCompile
Valid values are none, low, medium, and high.
Default value is high.

• BoundaryCompile
Valid values are none, low, medium, high, and top.
Default value is top.

The following attributes modify the compile strategy for each compile step:

• OptimizationPriorities (all modes)
Valid values are area, area_timing, timing, and timing_area.
Default value is timing_area.

```
• CanFlatten (full compile mode only)
Valid values are true and false.
Default value is false.


• CompileVerify (full and incremental compile modes only)
Valid values are true and false.
Default value is false.


• HasArithmetic (full compile mode only)
Valid values are true and false.
Default value is false.
(Deprecated: You will get better QoR for datapathes with just UltraOptimization ena
bled.)


• PartitionDP (full compile mode only)
Valid values are duplicate, dont_split and false.
Default value is false.
(Deprecated: You will get better QoR for datapathes with just UltraOptimization ena
bled.)


• AutoUngroup (full compile mode only)
This is a composite attribute with one string value with the
syntax: <mode> [<numCells>]
Valid values for the required mode are area, delay and false.
Default value is false.
Valid values for the optional numCells are integer numbers >0.
Default value is unset.


• MaxArea (full and incremental compile modes only)
Valid values are floating point numbers greater than or equal to 0.0.
Default value is 0.0.


• PreserveBoundaries (full and incremental compile modes only)
Valid values are true and false.
Default value is true.


• TestReadyCompile (full and incremental compile modes only)
Valid values are true and false.
Default value is false.


• UltraOptimization (all compile modes)
Valid values are true and false.
Default value is false.
```

For details about the impact of each attribute on the compile script, see the *Automated Chip Synthesis User Guide*.

SEE ALSO

```
compile(2)
get_license(2)
remove_license(2)
set_cost_priority(2)
set_flatten(2)
set_max_area(2)
set_minimize_tree_delay(2)
set_resource_allocation(2)
set_resource_implementation(2)
set_structure(2)
transform_csa(2)
uniquify(2)
write_compile_script(2)
compile_auto_ungroup_area_num_cells(3)
compile_auto_ungroup_delay_num_cells(3)
compile_auto_ungroup_override_wlm(3)
compile_implementation_selection(3)
compile_limit_down_sizing(3)
compile_map_low_drive(3)
compile_new_Boolean_structure(3)
compile_top_all_paths(3)
current_design(3)
```

acs_compile_script_suffix

Specifies the default suffix for script files generated by the **write_compile_script** command, sourced in the makefile generated by the **write_makefile** command, and located by the **report_pass_data** command.

TYPE

string

DEFAULT

autoscr

GROUP

acs_variables

DESCRIPTION

Specifies the default suffix for script files generated by the **write_compile_script** command, sourced in the makefile generated by the **write_makefile** command, and located by the **report_pass_data** command. For use in dc_shell-t (Tcl mode of dc_shell) only.

The script is placed in a file called pass*n*/scripts/*design*.*suffix*, where *suffix* is the value of the **acs_compile_script_suffix** variable.

To determine the current value of this variable, type **printvar acs_compile_script_suffix**.

SEE ALSO

report_pass_data(2)
write_compile_script(2)
write_makefile(2)

acs_constraint_file_suffix

Specifies the default suffix for constraint files generated by **write_partition_constraints** during the automated compile process.

TYPE

string

DEFAULT

con

GROUP

acs_variables

DESCRIPTION

Specifies the default suffix for constraint files generated by **write_partition_constraints** during the automated compile process. For use in dc_shell-t (Tcl mode of dc_shell) only.

The constraints are placed in a file called pass*n*/constraints/*design*.*suffix*, where *suffix* is the value of the **acs_constraint_file_suffix** variable.

To determine the current value of this variable, type **printvar acs_constraint_file_suffix**.

SEE ALSO

write_partition_constraints(2)

acs_cstr_report_suffix

Specifies the default suffix for constraint reports generated during the automated compile process. For use in dc_shell-t (Tcl mode of dc_shell) only.

TYPE

string

DEFAULT

cstr

GROUP

acs_variables

DESCRIPTION

Specifies the default suffix for constraint reports generated during the automated compile process. For use in dc_shell-t (Tcl mode of dc_shell) only.

The constraint report is placed in a file called pass*n*/reports/*design*.*suffix*, where *suffix* is the value of the **acs_cstr_report_suffix** variable.

To determine the current value of this variable, type **printvar acs_cstr_report_suffix**.

SEE ALSO

compile (2).

acs_db_suffix

Specifies the default suffix for .db files that are read or written during the automated compile process. For use in dc_shell-t (Tcl mode of dc_shell) only.

TYPE

string

DEFAULT

db

GROUP

acs_variables

DESCRIPTION

Specifies the default suffix for .db files that are read or written during the automated compile process. For use in dc_shell-t (Tcl mode of dc_shell) only.

The .db is placed in files called pass*n*/db/pre_compile/*design*.*suffix* and pass*n*/db/post_compile/*design*.*suffix*, where *suffix* is the value of the **acs_db_suffix** variable.

To determine the current value of this variable, type **printvar acs_db_suffix**.

SEE ALSO

read_partition(2), report_pass_data(2), write_compile_script(2), write_makefile(2), write_partition(2).

acs_dc_exec

Specifies the location of the dc_shell executable. This variable is used by the **acs_compile_design**, **acs_refine_design**, and **acs_recompile_design** commands to generate the makefile.

TYPE

string

DEFAULT

""

GROUP

acs_variables

DESCRIPTION

Specifies the location of the dc_shell executable. This variable is used by the **acs_compile_design**, **acs_refine_design**, and **acs_recompile_design** commands to generate the makefile. For use in dc_shell-t (Tcl mode of dc_shell) only.

To determine the current value of this variable, type **printvar acs_dc_exec**.

SEE ALSO

acs_compile_design(2)
acs_refine_design(2)
acs_recompile_design(2)

acs_default_pass_name

Specifies the prefix for the default data directory names. The pass number (either 0 or 1) is added to the prefix to generate the directory name.

TYPE

string

DEFAULT

pass

GROUP

acs_variables

DESCRIPTION

This variable specifies the prefix for the default data directory names. The pass number (0 when running acs_compile_design or 1 when running acs_refine_design) is added to the prefix to generate the directory name.

This variable has an effect only when running the acs_compile_design or acs_refine_design command without the -destination option.

To determine the current value of this variable, enter the following command:

```
      dc_shell-t> printvar acs_default_pass_name
```

SEE ALSO

acs_compile_design(2)
acs_refine_design(2)

acs_dir

This variable maps ACS directories to file types. This variable is used by the **acs_compile_design**, **acs_refine_design**, and **acs_recompile_design** commands to generate the ACS directory structure.

TYPE

array

GROUP

acs_variables

DESCRIPTION

This variable maps ACS directories to file types. This variable is used by the **acs_compile_design**, **acs_refine_design**, and **acs_recompile_design** commands to generate the ACS directory structure. For use in dc_shell-t (Tcl mode of dc_shell) only. This is a system variable only.

SEE ALSO

acs_compile_design(2)
acs_refine_design(2)
acs_recompile_design(2)

acs_exclude_extensions

Specifies the file endings of files you do not want the **acs_read_hdl** command to analyze.

TYPE

Tcl list of strings

DEFAULT

""

GROUP

acs_variables

DESCRIPTION

Specifies the file endings of files you do not want the **acs_read_hdl** command to analyze. This Tcl variable is needed only if the language extensions variable (**acs_verilog_extensions**, **acs_vhdl_extensions**, or both) is set to an empty list ({}), in which case all files are read, including files in the source directory that are not supposed to be analyzed. You can use **acs_exclude_extensions** to exclude files with certain extensions.

Some examples are

```
      set acs_exclude_extensions { .inc .v_in .def }

      set acs_exclude_extensions { .h .H .hh .c .C .cc }

      set acs_exclude_extensions { .txt .doc }
```

To determine the current value of this variable, enter the following command:

```
      dc_shell-t> printvar acs_exclude_extensions
```

SEE ALSO

acs_read_hdl(2)
acs_exclude_list(3)
acs_hdl_source(3)
acs_variables(3)
acs_verilog_extensions(3)
acs_vhdl_extensions(3)

acs_exclude_list

Specifies files and directories you do not want the **acs_read_hdl** command to analyze.

TYPE

Tcl list of strings

DEFAULT

"[list $synopsys_root]"

GROUP

acs_variables

DESCRIPTION

Specifies files and directories you do not want the **acs_read_hdl** command to analyze. Set the value of this Tcl variable to a list of directory and file names that you do not want the **acs_read_hdl** command to analyze. When the **acs_read_hdl** command is expanding wildcards or traversing directories, it will analyze a file only if it is not matched by any item of this list. A file matches an item on the list if the list specifies the file, its directory, or one of its parent directories. Thus, if you specify a directory name in the list, all of its subdirectories are excluded also.

Specify all files and directories with their full paths. If you give a filename without a path, all files with that name are excluded, regardless of where they are located. Additionally, the Tcl UNIX-like "file-globbing" features are valid; you can use the wildcards *, ?, and [].

To determine the current value of this variable, type

```
      dc_shell-t> printvar acs_exclude_list
```

The recommended way to set this variable is to always add items instead of setting it directly:

```
      dc_shell-t> set acs_exclude_list \
             [concat $acs_exclude_list "/new/path/to/be/excluded"]
```

SEE ALSO

acs_read_hdl(2)
acs_exclude_extensions(3)
acs_hdl_source(3)
acs_variables(3)
acs_verilog_extensions(3)
acs_vhdl_extensions(3)

acs_global_user_compile_strategy_script

Specifies the base file name for the user-defined default compile strategy. For use in dc_shell-t (Tcl mode of dc_shell) only.

TYPE

string

DEFAULT

default

GROUP

acs_variables

DESCRIPTION

Specifies the base file name for the user-defined default compile strategy. For use in dc_shell-t (Tcl mode of dc_shell) only. A compile strategy is a script that includes commands to set the **compile** variables, set the **compile** attributes, and run the **compile** command.

Before generating a partition compile script, Automated Chip Synthesis looks for a user-defined partition compile strategy in the file scripts/pass*n*/*partition_name*.*suffix*. If a partition-specific compile strategy does not exist, Automated Chip Synthesis looks for a user-defined default compile strategy in the file scripts/pass*n*/$acs_global_user_compile_strategy_script.*suffix*. If a user-defined compile strategy exists, Automated Chip Synthesis uses it instead of the default compile strategy when generating the compile script.

To determine the current value of this variable, type **printvar acs_global_user_compile_strategy_script**.

SEE ALSO

acs_compile_design (2), **acs_refine_design** (2), **acs_recompile_design** (2), **write_compile_script** (2), **acs_user_compile_strategy_script_suffix** (3).

acs_hdl_source

Specifies the location of the source code files analyzed by the **acs_read_hdl** command.

TYPE

Tcl list of strings

DEFAULT

""

GROUP

acs_variables

DESCRIPTION

Specifies the location of the source code files analyzed by the **acs_read_hdl** command. Set the value of this Tcl variable to a list of file and directory names to specify where your HDL source code files are located. The **acs_read_hdl** command expands wildcards in that list (in a "glob" Tcl-command style manner) and analyzes all files from the list and from the specified directories, as directed by the other variables listed in the SEE ALSO section.

If this variable is set to a value other than an empty list, it prevents the **acs_read_hdl** command from using the search_path as HDL source list. The value of the **acs_hdl_source** variable is overridden by the **-hdl_source** option of the command.

To determine the current value of this variable, type

```
      dc_shell-t> printvar acs_hdl_source
```

SEE ALSO

acs_read_hdl(2)
acs_exclude_extensions(3)
acs_exclude_list(3)
acs_variables(3)
acs_verilog_extensions(3)
acs_vhdl_extensions(3)

acs_hdl_verilog_define_list

Specifies a list of text macros which are to be defined by the **acs_read_hdl** command during verilog file analysis.

TYPE

Tcl list of strings

DEFAULT

""

GROUP

acs_variables

DESCRIPTION

Specifies a list of text macros which are to be defined by the **acs_read_hdl** command during verilog file analysis.

Set the value of this Tcl variable to a list of verilog text macro identifiers. While analyzing any verilog source file all of these text macros will be defined, influencing any existing conditional verilog preprocessor directives.

For **acs_read_hdl** to achieve the same effect as with

```
      dc_shell-t> analyze -define compileForSynthesis ...
```

use

```
      dc_shell-t> set acs_hdl_verilog_define_list compileForSynthesis
      dc_shell-t> acs_read_hdl ...
```

To determine the current value of this variable, type

```
      dc_shell-t> printvar acs_hdl_source
```

SEE ALSO

acs_read_hdl(2)
analyze(2)

acs_lic_wait

Specifies the maximum wait time for checking out all the licenses required by a compile job.

TYPE

integer

DEFAULT

0

GROUP

acs_variables

DESCRIPTION

By default (acs_lic_wait = 0), the makefile generated by Automated Chip Synthesis runs the dc_shell command without the "-wait #" option. To ensure that all the needed features and licenses are checked out before you run the compile job, set **acs_lic_wait** to the maximum wait time (unit is minute).

This variable is used only by the **acs_compile_design**, **acs_refine_design**, and **acs_recompile_design** commands.

To determine the current value of this variable, type **printvar acs_lic_wait**.

SEE ALSO

acs_compile_design(2)
acs_recompile_design(2)
acs_refine_design(2)

acs_log_file_suffix

Specifies the default suffix for log files generated during the automated compile process. For use in dc_shell-t (Tcl mode of dc_shell) only.

TYPE

string

DEFAULT

log

GROUP

acs_variables

DESCRIPTION

Specifies the default suffix for log files generated during the automated compile process. For use in dc_shell-t (Tcl mode of dc_shell) only.

The log is placed in a file called pass*n*/logs/*design*.*suffix*, where *suffix* is the value of the **acs_log_file_suffix** variable.

To determine the current value of this variable, type **printvar acs_log_file_suffix**.

SEE ALSO

write_makefile(2).

acs_make_args

Specifies the command arguments for the make utility command (gmake, by default).

TYPE

string

DEFAULT

set acs_make_args

GROUP

acs_variables

DESCRIPTION

This variable specifies the command arguments for the make utility command. The default make utility command is gmake. Ensure that this variable specified valid arguments for the current make utility command.

To determine the current value of this variable, enter the following command:

```
      dc_shell-t> printvar acs_make_args
```

SEE ALSO

acs_compile_design(2)
acs_recompile_design(2)
acs_refine_design(2)
acs_make_exec(3)

acs_make_exec

Specifies the make utility command used to run the compile jobs.

TYPE

string

DEFAULT

gmake

GROUP

acs_variables

DESCRIPTION

This variable specifies the make utility command used to run the compile jobs.

To determine the current value of this variable, enter the following command:

```
      dc_shell-t> printvar acs_make_exec
```

SEE ALSO

acs_compile_design(2)
acs_recompile_design(2)
acs_refine_design(2)
acs_make_args(3)

acs_makefile_name

Specifies the file name for the makefile generated by the **write_makefile** command and run by the **compile_partitions** command. For use in dc_shell-t (Tcl mode of dc_shell) only.

TYPE

string

DEFAULT

makefile

GROUP

acs_variables

DESCRIPTION

Specifies the filename for the makefile generated by the **write_makefile** command and run by the **compile_partitions** command. For use in dc_shell-t (Tcl mode of dc_shell) only.

To determine the current value of this variable, type **printvar acs_makefile_name**.

SEE ALSO

compile_partitions (2), **write_makefile** (2).

acs_num_parallel_jobs

Specifies the number of compile jobs to run in parallel when using **gmake** as the **make** utility.

TYPE

integer

DEFAULT

1

GROUP

acs_variables

DESCRIPTION

Specifies the number of compile jobs to run in parallel when using **gmake** as the **make** utility. For use in dc_shell-t (Tcl mode of dc_shell) only.

This variable is used only by the **acs_compile_design**, **acs_refine_design**, and **acs_recompile_design** commands.

To determine the current value of this variable, type **printvar acs_num_parallel_jobs**.

SEE ALSO

acs_compile_design(2)
acs_refine_design(2)
acs_recompile_design(2)

acs_override_report_suffix

Specifies the suffix for user-defined partition report scripts.

TYPE

string

DEFAULT

report

GROUP

acs_variables

DESCRIPTION

Specifies the suffix for user-defined partition report scripts. For use in dc_shell-t (Tcl mode of dc_shell) only.

In generating a partition compile script with the **write_compile_script** command, Automated Chip Synthesis looks for a user-defined partition report script in scripts/pass*n*/*partition_name*.*suffix*. If a user-defined report script exists, Automated Chip Synthesis uses it instead of generating the report with its default process.

To determine the current value of this variable, type **printvar acs_override_report_suffix**.

SEE ALSO

acs_compile_design(2)
acs_recompile_design(2)
acs_refine_design(2)
write_compile_script(2)
acs_area_report_suffix(3)
acs_cstr_report_suffix(3)
acs_qor_report_suffix(3)
acs_timing_report_suffix(3)

acs_override_script_suffix

Specifies the suffix for user-defined partition compile scripts. For use in dc_shell-t (Tcl mode of dc_shell) only.

TYPE

string

DEFAULT

scr

GROUP

acs_variables

DESCRIPTION

Specifies the suffix for user-defined partition compile scripts. For use in dc_shell-t (Tcl mode of dc_shell) only.

Before generating a partition compile script, Automated Chip Synthesis looks for a user-defined partition compile script in scripts/pass*n*/*partition_name*.*suffix*. If a user-defined script exists, Automated Chip Synthesis uses it instead of generating a default script.

To determine the current value of this variable, type **printvar acs_override_script_suffix**.

SEE ALSO

acs_compile_design (2), **acs_refine_design** (2), **acs_recompile_design** (2), **write_compile_script** (2).

acs_qor_report_suffix

Specifies the suffix for QOR reports generated during the automated compile process; the default is *qor*. For use in dc_shell-t (Tcl mode of dc_shell) only.

TYPE

string

DEFAULT

qor

GROUP

acs_variables

DESCRIPTION

Specifies the suffix for QOR reports generated during the automated compile process; the default is *qor*. For use in dc_shell-t (Tcl mode of dc_shell) only.

Unless you specify the **-no_reports option**, the compile scripts generated by the **write_compile_script** command includes a command to generate a QOR report. The QOR report is placed in a file called pass*n*/reports/*design*.*suffix*, where *suffix* is the value of the **acs_qor_report_suffix** variable.

To determine the current value of this variable, type **printvar acs_qor_report_suffix**.

SEE ALSO

write_compile_script (2).

acs_svf_suffix

Specifies the default suffix for svf files that are written during the automated compile process.

TYPE

string

DEFAULT

svf

GROUP

acs_variables

DESCRIPTION

Specifies the default suffix for svf files that are written during the automated compile process. For use in dc_shell-t (Tcl mode of dc_shell) only.

To determine the current value of this variable, type **printvar acs_svf_suffix**.

SEE ALSO

set_svf(2)

acs_timing_report_suffix

Specifies the suffix for timing reports generated during the automated compile process. For use in dc_shell-t (Tcl mode of dc_shell) only.

TYPE

string

DEFAULT

tim

GROUP

acs_variables

DESCRIPTION

Specifies the suffix for timing reports generated during the automated compile process. For use in dc_shell-t (Tcl mode of dc_shell) only.

Unless you specify the **-no_reports** option, the compile scripts generated by the **write_compile_script** command include a command to generate a timing report. The timing report is placed in a file called pass*n*/reports/*design*.*suffix*, where *suffix* is the value of the **acs_timing_report_suffix** variable.

To determine the current value of this variable, type **printvar acs_timing_report_suffix**.

SEE ALSO

write_compile_script (2).

acs_use_autopartition

Sets autopartitioning as the default partitioning strategy for the chip-level compile commands.

TYPE

Boolean

DEFAULT

false

GROUP

acs_variables

DESCRIPTION

By default (acs_use_autopartition = false), the chip-level compile commands partition the design by selecting the first-level subdesigns and reference designs of multiple instances as the compile partitions. When this variable is true, the chip-level compile commands determine the compile partitions based on size estimation and connectivity.

To determine the current value of this variable, enter **printvar acs_use_autopartition**.

SEE ALSO

acs_compile_design(2)
acs_refine_design(2)
acs_recompile_design(2)
report_partitions(2)

acs_use_default_delays

Sets top-down environment propagation as the constraint generation method for GTECH designs (the acs_compile_design command). When this variable is false (the default), the acs_compile_design command uses RTL budgeting to generate constraints for the compile partitions.

TYPE

Boolean

DEFAULT

false

GROUP

acs_variables

DESCRIPTION

This variable sets top-down environment propagation as the constraint generation method for GTECH designs (the acs_compile_design command). When this variable is false (the default), the acs_compile_design command uses RTL budgeting to generate constraints for the compile partitions.

To determine the current value of this variable, enter the following command:

```
      dc_shell-t> printvar acs_use_default_delays
```

SEE ALSO

acs_compile_design(2)

acs_user_budgeting_script

Specifies the file name for the user-defined budgeting script.

TYPE

string

DEFAULT

budget.scr

GROUP

acs_variables

DESCRIPTION

Specifies the file name for the user-defined budgeting script. For use in dc_shell-t (Tcl mode of dc_shell) only.

Before performing design budgeting, Automated Chip Synthesis looks for a user-defined budgeting script in the file scripts/pass*n*/$acs_user_budgeting_script. If a user-defined script exists, Automated Chip Synthesis uses it to perform design budgeting, instead of using the default budgeting script.

To determine the current value of this variable, type **printvar acs_user_budgeting_script**.

SEE ALSO

acs_compile_design(2)
acs_refine_design(2)
acs_recompile_design(2)

acs_user_compile_strategy_script_suffix

Specifies the suffix for user-defined partition compile strategies. For use in dc_shell-t (Tcl mode of dc_shell) only.

TYPE

string

DEFAULT

compile

GROUP

acs_variables

DESCRIPTION

Specifies the suffix for user-defined partition compile strategies. For use in dc_shell-t (Tcl mode of dc_shell) only.

A compile strategy is a script that includes commands to set the **compile** variables, set the **compile** attributes, and run the **compile** command.

Before generating a partition compile script, Automated Chip Synthesis looks for a user-defined partition compile strategy in scripts/pass*n*/*partition_name*.*suffix*. If a user-defined compile strategy exists, Automated Chip Synthesis uses it when generating the compile script, instead of using the default compile strategy.

To determine the current value of this variable, type **printvar acs_user_compile_strategy_script_suffix**.

SEE ALSO

acs_compile_design (2), **acs_refine_design** (2), **acs_recompile_design** (2), **write_compile_script** (2).

acs_variables

Specify various Automated Chip Synthesis controls.

SYNTAX

```
string acs_dc_exec           $SYNOPSYS/sparcOS5/syn/bin/dc_shell
list   acs_exclude_extensions          {}
list   acs_exclude_list                {}
list   acs_hdl_source                  {}
string acs_num_parallel_jobs           1
string acs_user_budgeting_script       "budget.scr"
string acs_user_compile_strategy_script_suffix        "compile"
list   acs_user_dir(path)              {}
list   acs_verilog_extensions          {".v"}
list   acs_vhdl_extensions             {".vhd"}
string acs_work_dir                    "[pwd]"
list   check_error_list                {}
string ilm_preserve_core_constraints   "false"
string acs_read_hdl_use_read_file_for_vhdl   "true"
```

DESCRIPTION

These variables directly affect Automated Chip Synthesis and are used only in dc_shell-t (Tcl mode of dc_shell).

To view this man page online from within dc_shell-t, type **man acs_variables**. To view an individual variable description, type **man *var***, where *var* is the name of the variable. To determine the current value of a variable, type **printvar** *var*.

acs_area_report_suffix
: Specifies the suffix for area reports generated during the automated compile process. The default is area.

acs_autopart_max_area
: Used with other acs variables to control chip-level partitioning. Defines partition threshold. For use in dc_shell-t (Tcl mode of dc_shell) only.

acs_autopart_max_percent
: Used with other acs variables to control chip-level partitioning. For use in dc_shell-t (Tcl mode of dc_shell) only.

acs_use_autopartition
: Used with other acs variables to control chip-level partitioning. If the variable is true, the design will be partitioned automatically. For use in dc_shell-t (Tcl mode of dc_shell) only.

acs_budgeted_cstr_suffix
: Specifies the suffix for constraint files generated by the **derive_partition_budgets** command. The default is con.

acs_compile_script_suffix
: Specifies the default suffix for script files generated by the

write_compile_script command, sourced in the makefile generated by the **write_makefile** command, and located by the **report_pass_data** command. The default is autoscr.

acs_constraint_file_suffix
: Specifies the default suffix for constraint files generated during the automated compile process. The default is con.

acs_cstr_report_suffix
: Specifies the default suffix for constraint reports generated during the automated compile process. The default is cstr.

acs_db_suffix
: Specifies the default suffix for .db files that are read or written during the automated compile process. The default is db.

acs_dc_exe
: Specifies the location of the dc_shell executable. This variable is used by the **acs_compile_design**, **acs_refine_design**, and **acs_recompile_design** commands to generate the makefile. The default is $SYNOPSYS/$arch/syn/bin/dc_shell.

acs_exclude_extensions
: Specifies the file endings of files that will not be analyzed. When the **acs_read_hdl** command is collecting source files from expanded wildcards or directories, it takes only files that do not end with one of the strings from this list.

acs_exclude_list
: Specifies files and directories that will not be analyzed. When the **acs_read_hdl** command is expanding wildcards or traversing directories it will analyze a file only if it is not matched by any item of this list.

acs_global_user_compile_strategy_script
: Specifies the file name for the user-defined default compile strategy. The default is default.compile.

acs_hdl_source
: Specifies the location of the HDL source code files. The **acs_read_hdl** command expands wildcards in that list (in a Tcl-command "glob"-style manner) and analyzes all files from the list and from the specified directories.

acs_log_file_suffix
: Specifies the default suffix for log files generated during the automated compile process. The default is log.

acs_makefile_name
: Specifies the filename for the makefile generated by the **write_makefile** command and run by the **compile_partitions** command. The default is Makefile.

acs_num_parallel_jobs
: Specifies the number of compile jobs to run in parallel when using **gmake** as the **make** utility. The default is 1.

acs_override_script_suffix
: Specifies the suffix for user-defined partition compile scripts. The default

is scr.

acs_qor_report_suffix
Specifies the suffix for QOR reports generated during the automated compile process. The default is qor.

acs_timing_report_suffix
Specifies the suffix for timing reports generated during the automated compile process. The default is tim.

acs_lic_wait
Specifies the maximum wait time for checking out all the licenses required by a compile job. By default (acs_lic_wait = 0), the makefile generated by Automated Chip Synthesis runs the dc_shell command without "-wait #" option. If you want to ensure that all the needed features/licenses are checked out before running the compile job, set acs_lic_wait to the maximum wait time (unit is minute).

acs_user_budgeting_script
Specifies the file name for the user-defined budgeting script. The default is budget.scr.

acs_user_compile_strategy_script_suffix
Specifies the suffix for user-defined partition compile strategies. The default is compile.

acs_user_dir(path)
Specifies a customized directory structure for Automated Chip Synthesis. For use in .synopsys_dc.setup of dc_shell-t (Tcl mode of dc_shell) only.

acs_override_report_suffix
Specifies the suffix for user-defined partition report scripts. The default is report.

acs_verilog_extensions
Specifies the file endings of Verilog files. The default value is {".v"}. When the **acs_read_hdl** command is collecting Verilog source files from expanded wildcards or directories, it takes only files that end with one of the strings from this list.

acs_vhdl_extensions
Specifies the file endings of VHDL files. When the **acs_read_hdl** command is collecting VHDL source files from expanded wildcards or directories, it takes only files that end with one of the strings from this list.

acs_work_dir
Specifies the root of the Automated Chip Synthesis project directory. During startup, this variable is set to the current working directory. This value is used when locating input files for Automated Chip Synthesis commands and when generating absolute path names.

check_error_list
Specifies a list of error codes for which messages are to be checked during the current dc_shell session.

ilm_preserve_core_constraints

Specifies if ilm_mode should be turned on (value "true") or off ("false"). The ilm_mode, if turned on during constraint setting, helps characterizing an instance of an ILM and creating a constraint file for it that contains the constraints for the register-to-register logic of the design. The default value is "false", turning ilm_mode off.

acs_read_hdl_use_read_file_for_vhdl

If set to true, then **acs_read_hdl -[auto_]update** command uses the **read_file** command to read VHDL files. Otherwise, it uses the **analyze** and **elaborate -update** commands to read VHDL files.

SEE ALSO

acs_compile_design(2)
acs_read_hdl(2)
acs_recompile_design(2)
acs_refine_design(2)
check_error(2)
compile(2)
compile_partitions(2)
create_pass_directories(2),
extract_ilm(2)
read_partition(2),
remove_pass_directories(2),
report_pass_data(2),
write_compile_script(2),
write_makefile(2),
write_partition(2),
write_partition_constraints(2)

acs_verilog_extensions

Specifies the file endings of files analyzed as Verilog source code by the **acs_read_hdl** command.

TYPE

Tcl list of strings

DEFAULT

".v"

GROUP

acs_variables

DESCRIPTION

Specifies the file endings of files analyzed as Verilog source code by the **acs_read_hdl** command. Set the value of this Tcl variable to a list of file extensions to specify the filename endings of your Verilog source code files. When the **acs_read_hdl** command is collecting Verilog source files from the specified source code location (that is, if the **-format** option is not set to "vhdl"), it will analyze as Verilog files all files that end with one of the strings from this list. If the variable is not defined or is set to an empty list, all files in the source code location are collected and considered Verilog source code.

To determine the current value of this variable, type

```
      dc_shell-t> printvar acs_verilog_extensions
```

SEE ALSO

acs_read_hdl(2)
acs_exclude_extensions(3)
acs_exclude_list(3)
acs_hdl_source(3)
acs_variables(3)
acs_vhdl_extensions(3)

acs_vhdl_extensions

Specifies the file endings of files the **acs_read_hdl** command analyzes as VHDL source code.

TYPE

list

DEFAULT

".vhd"

GROUP

acs_variables

DESCRIPTION

Set the value of this variable to a list of file extensions that specify the file name endings of your VHDL source code files. When the **acs_read_hdl** command is collecting VHDL source files from the specified source code location (that is, if the **-format** option of the command is *not* set as verilog), the tool analyzes as VHDL files all files that end with one of the strings you define for this list.

If you do not define the **acs_vhdl_extensions** variable, or if you set the value to an empty list, the tool collects all files in the source code location and considers them to be VHDL source code.

To see the current value of this variable, type

```
      psyn_shell-xg-t> printvar acs_vhdl_extensions
```

SEE ALSO

acs_read_hdl(2)
acs_exclude_extensions(3)
acs_exclude_list(3)
acs_hdl_source(3)
acs_variables(3)
acs_verilog_extensions(3)

acs_work_dir

Specifies the root of the Automated Chip Synthesis project directory. For use in dc_shell-t (Tcl mode of dc_shell) only.

TYPE

string

DEFAULT

the current working directory

GROUP

acs_variables

DESCRIPTION

Specifies the root of the Automated Chip Synthesis project directory. During startup, this variable is set to the current working directory. For use in dc_shell-t (Tcl mode of dc_shell) only.

This value is used when locating input files for Automated Chip Synthesis commands and when generating absolute path names.

To determine the current value of this variable, type **printvar acs_work_dir**.

Note: Character @ is reserved to specify pass dependent directory names. Its use as a literal character in ACS directory paths is not allowed.

SEE ALSO

create_pass_directories (2), **read_partition** (2), **remove_pass_directories** (2), **report_pass_data** (2), **write_compile_script** (2), **write_makefile** (2), **write_partition** (2), **write_partition_constraints** (2).

alib_library_analysis_path

Specifies a single path, similar to a search path, for reading and writing the alib files that correspond to the target libraries.

TYPE

string

DEFAULT

"./"

DESCRIPTION

The tool loads alibs from under this path during compile. The tool stores alibs to this path when the *alib_analyze_libs* command is issued.

SEE ALSO

attributes

Lists the predefined Synopsys attributes.

DESCRIPTION

Attributes are properties assigned to objects such as nets, cells, and clocks, and describe design features to be considered during optimization.

Attributes are grouped into the following categories:

- cell
- clock
- design
- library cell
- net
- pin
- port
- read-only
- reference

Definitions for these attributes are provided in the subsections that follow.

There are a number of commands used to set attributes, however, most attributes can be set with the **set_attribute** command. If the attribute definition specifies a **set** command, use it to set the attribute. Otherwise, use **set_attribute**.

Some attributes are informational, or "read-only." You cannot set the value of these attributes. Most attribute groups contain read-only attributes; however, a complete list of these attributes is provided in the "Read Only" subsection.

Some attributes are "instance-specific", meaning they can be applied to specified objects in the design hierarchy. The following is a list of these attributes:

Certain attributes are specific to Power Compiler objects. For information on the Power Compiler attributes, refer to the **power_attributes** manual page.

- disable_timing
- load
- test_assume

To determine the value of an attribute, use the **get_attribute** command. To remove attributes, use the **remove_attribute** command.

For a more detailed explanation of attributes, refer to the man pages of the appropriate **set** command.

Note: Path groups, cell delay, net delay, external delay, point-to-point timing specification, and arrival information are not represented as attributes, therefore cannot be manipulated with **attributes** commands.

Cell Attributes

area

Specifies the area of a cell. This attribute doesn't exist on a hierarchical cell.
We calculate the attribute by the cell's boundary points.
This attribute is "read-only" and cannot be set by the user.

aspect_ratio

Specifies **height**:**width** ratio of a cell. This attribute doesn't exist on a hierarchical cell.
This attribute is "read-only" and cannot be set by the user.

async_set_reset_q

Establishes the value (0 or 1) that should be assigned to the q output of an inferred register if set and reset are both active at the same time. To be used with **async_set_reset_qn**. Use these attributes only if you have used the **one_hot** or **one_cold** attributes/directives in your HDL description *and* your technology library is written using pre-V3.0a syntax; *or* if your technology library does not use a consistent convention for q and qn when set and reset are both active. If a V3.0a or later syntax technology library is used, then by default if set and reset are both active at the same time Design Compiler will use the convention of the selected technology library (**target_library**). Set with **set_attribute**.
Note: If you are unsure whether or not your technology library uses V3.0a syntax, ask your ASIC vendor.

async_set_reset_qn

Establishes the value (0 or 1) that should be assigned to the qn output of an inferred register if set and reset are both active at the same time. To be used with **async_set_reset_q**. Use these attributes only if you have used the **one_hot** or **one_cold** attributes/directives in your HDL description *and* your technology library is written using pre-V3.0a syntax; *or* if your technology library does not use a consistent convention for q and qn when set and reset are both active. If a V3.0a or later syntax technology library is used, then by default if set and reset are both active at the same time Design Compiler will use the convention of the selected technology library (**target_library**). Set with **set_attribute**.
Note: If you are unsure whether or not your technology library uses V3.0a syntax, ask your ASIC vendor.

combinational_type_exact

Specifies the replacement gate to use for cells specified in the cell list. Compile attempts to convert combinational gates tagged with **set_compile_type** to the specified replacement combinational gate. Set with **set_combinational_type**.

disable_timing

Disables the timing arcs of a cell. This has the same effect on timing as not having the arc in the library. Set with **set_disable_timing.**

dont_touch

Identifies cells to be excluded from optimization. Values are *true* (the default) or *false*. Cells with the **dont_touch** attribute set to *true* are not modified or replaced during **compile**. Setting **dont_touch** on a hierarchical cell sets the attribute on all cells below it. Set with **set_dont_touch** .

fall_delay

Specifies an offset from the falling edge of the ideal clock waveform. Affects all clock pins on this cell. Set with **set_clock_skew -fall_delay**.

flip_flop_type

Stores the name of the specified flip-flop to be converted from the **target_library**. **compile** automatically converts all tagged flip-flops to the specified (or one similar) type. Set with **set_register_type -flip_flop** *flip_flop_name [cell_list]* .

flip_flop_type_exact

Stores the name of the specified flip-flop to be converted from the **target_library**. **compile** automatically converts all tagged flip-flops to the exact flip-flop type. Set with **set_register_type -exact -flip_flop** *flip_flop_name [cell_list]* .

height

Specifies the height of a cell. This attribute doesn‚Äôt exist on a hierarchical cell.
We use the cell’s cell boundary to calculate its height.
This attribute is "read-only" and cannot be set by the user.

is_black_box

true if the cell’s reference is not linked to a design or is linked to a design that doesn’t have a functionality. This attribute is "read-only" and cannot be set by the user.

is_combinational

true if all cells of a design and all designs in its hierarchy are combinational. A cell is combinational if it is non-sequential or non-tristate and all of its outputs compute a combinational logic function. The **report_lib** command will report such a cell as not a black-box. This attribute is *read-only* and cannot be set by the user.

is_dw_subblock

true if the object (a cell, a reference, or a design) is a DW subblock that was automatically elaborated. This attribute is "read-only" and cannot be set by the user.
NOTE: DW subblocks that are manually elaborated will not have this attribute.

is_hierarchical

true if the design is not a leaf design (for example, not from a technology library). This attribute is "read-only" and cannot be set by the user.

is_mapped

true if the cell is not generic logic. This attribute is "read-only" and cannot be set by the user.

is_sequential

true if the cell is sequential. A cell is sequential if it is not combinational. This attribute is "read-only" and cannot be set by the user.

is_synlib_module
true if the object (a cell, a reference, or a design) refers to an unmapped module reference or if the object is (or refers to) a design that was automatically elaborated from a synlib module or a synlib operator. This attribute is "read-only" and cannot be set by the user.
NOTE: synlib modules that are manually elaborated will not have this attribute.

is_synlib_operator
true if the object (a cell or a reference) is a synthetic library operator reference. This attribute is "read-only" and cannot be set by the user.

is_test_circuitry
Set by **insert_dft** on the scan cells and nets added to a design during the addition of test circuitry. This attribute is "read-only" and cannot be set by the user.

is_unmapped
true if the cell is generic logic. This attribute is "read-only" and cannot be set by the user.

latch_type_exact
Stores the name of the specified latch to be converted from the **target_library**. **compile** automatically converts all tagged latches to the exact latch type. Set with **set_register_type -latch** *latch_name [cell_list]* .

macro_area_percentage
Specifies the percentage of total macro area of a soft macro‚Äôs physical_area.
This attribute is "read-only" and cannot be set by the user.

map_only
When set to *true*, **compile** will attempt to map the object exactly in the target library, and will exclude the object from logic-level optimization (flattening and structuring). The default is *false*. Set with **set_map_only**.

max_fall_delay
A floating point value that specifies the maximum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

max_metal_layer
Specifies the reserved maximum metal layer name of a soft macro or a black box.
This attribute is "read-only" and cannot be set by the user.

max_rise_delay
A floating point value that specifies the maximum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

max_time_borrow
A floating point number that establishes an upper limit for time borrowing; that is, it prevents the use of the entire pulse width for level-sensitive latches. Units are those used in the technology library. Set with

set_max_time_borrow.

min_fall_delay
A floating point value that specifies the minimum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

min_metal_layer
Specifies the reserved minimum metal layer name of a soft macro or a black box.
This attribute is "read-only" and cannot be set by the user.

min_rise_delay
A floating point value that specifies the minimum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

minus_uncertainty
Specifies a negative uncertainty from the edges of the ideal clock waveform. Affects all clock pins on this cell. Set with **set_clock_skew -minus_uncertainty**.

number_of_black_box
Specifies the count of black boxes in a hierarchical cell or a soft macro.
Whether a cell is a black box can be determined by the attribute is_black_box.
This attribute is "read-only" and cannot be set by the user.

number_of_io_cell
Specifies the count of io cells in a hierarchical cell or a soft macro.
We count cells in when its mask_layout_type is either io_pad or corner_pad.
This attribute is "read-only" and cannot be set by the user.

number_of_macro
Specifies the count of macros in a hierarchical cell or a soft macro.
We count cells in when its mask_layout_type is macro.
This attribute is "read-only" and cannot be set by the user.

number_of_pinshape
Specifies the number of pin shapes of a soft macro.
This attribute is "read-only" and cannot be set by the user.

number_of_standard_cell
Specifies the count of standard cells in a hierarchical cell or a soft macro.
We count cells in when its mask_layout_type is std.
This attribute is "read-only" and cannot be set by the user.

physical_area
Specifies the physical area of a hierarchical cell or a soft macro.
The physical area of a hierarchical cell is the sum of its direct children's **physical_area** or **area**. If a direct child is a hierarchical cell, then **physical_area** is used, if a child is a standard cell or macro, then **area** is used, else that child is skipped.
The physical area of a soft macro is the sum of its children's **physical_area** or **area**. The children will be iterated from the sub-design file. If a child is a soft macro, then **physical_area** is used, if a child is a standard cell

or hard macro, **area** is used, otherwise that child is skipped. This attribute is "read-only" and cannot be set by the user.

physical_area_percentage_in_top_design
: Specifies the percentage of **physical_area** of a soft macro in the top design. This attribute is "read-only" and cannot be set by the user.

plus_uncertainty
: Specifies a positive uncertainty from the edges of the ideal clock waveform. Affects all clock pins on this cell. Set with **set_clock_skew -plus_uncertainty**.

propagated_clock
: Specifies that the clock edge times be delayed by propagating the values through the clock network. Affects all clock pins on this cell. If this attribute is not present, ideal clocking is assumed. Set with **set_clock_skew -propagated**.

ref_name
: The reference name of a cell. This attribute is "read-only" and cannot be set by the user.

full_name
: The hierarchical name of cell, pin or net. This attribute is "read-only" and cannot be set by the user.

rise_delay
: Specifies an offset from the rising edge of the ideal clock waveform. Affects all clock pins on this cell. Set with **set_clock_skew -rise_delay**.

scan
: When *true*, specifies that the cell is always replaced by an equivalent scan cell during **insert_dft**. When false, the cell is not replaced. Set with **set_scan** .

scan_chain
: Includes the specified cells of the referenced design in the scan-chain whose index is the value of this attribute. Set with **set_scan_chain** .

scan_element
: Determines if sequential cells in the specified designs are replaced by equivalent scan cells during **insert_scan**. When *true*, the default, **insert_scan** replaces cell_design_ref_list with equivalent scan cells. The scan cells are not replaced when set to *false*. Set with **set_scan_element**.

scan_latch_transparent
: When *true*, makes specified cells transparent during ATPG. For hierarchical cells, the effects apply hierarchically to level-sensitive leave cells. Set with **set_scan_transparent**. Remove with **remove_attribute**.

test_isolate
: Indicates that the specified sequential cells, pins, or ports are to be logically isolated and considered untestable during test design rule checking by **check_test**. When this attribute is set on a cell, it is also placed on all pins of that cell. Do not set this attribute on a hierarchical cell. Use

report_test -assertions for a report on isolated objects. Set with **set_test_isolate**.
Note: Setting this attribute suppresses the warning messages associated with the isolated objects.

test_routing_position
Specifies the preferred routing order of the scan-test signals of the identified cells. Set with **set_test_routing_order** .

ungroup
Removes a level of hierarchy by exploding the contents of the specified cell in the current design. If specified on a reference object, cells using that reference are ungrouped during **compile**. Set with **set_ungroup** .

xnf_init
A string that specifies the value of a storage element during power-on initialization and assertion of the global Set/Reset signal. For flip-flops, the two allowed string values are *S*, to define an inital Set condition; or , to define an initial Reset condition. For ROM symbols, the string value is a hex character string that defines the pattern of the ROM. Setting an **xnf_init** attribute on a cell causes the Synopsys XNF writer to create an INIT attribute on that cell in the XNF netlist; the value of this INIT attribute will match the value specified for the **xnf_init** attribute.
Set this attribute with **set_attribute**, after executing **replace_fpga** and before executing **write -f xnf**.
Note: Set this attribute only on cells that are xnf primitives; **xnfmerge** will not propagate the attribute down into any child cells. In addition, Do not set this attribute on D flip-flops if any of the asynchronous pins are being used.

xnf_loc
A string that specifies the Xilinx location of a cell. No checks are performed to verify that the specified location is valid; for valid location string values, refer to the Xilinx *XC4000 Databook*. Setting an **xnf_loc** attribute on a cell causes the Synopsys XNF writer to create a LOC attribute on that cell in the XNF netlist; the value of this LOC attribute will match the value specified for the **xnf_loc** attribute.
Set this attribute with **set_attribute**, after executing **replace_fpga** and before executing **write -f xnf**.
Note: Set this attribute only on cells that are xnf primitives; **xnfmerge** will not propagate the attribute down into any child cells.

Clock Attributes

clock_fall_transition
Sets the falling transition value on the specified clock list. The clock_fall_transition overrides the calculated transition times on clock pins of registers and associated nets. Set using **set_clock_transition**.

clock_rise_transition
Sets the rising transition value on the specified clock list. The clock_rise_transition overrides the calculated transition times on clock pins of registers and associated nets. Set using **set_clock_transition**.

dont_touch_network

When a design is optimized, **compile** assigns **dont_touch** attributes to all cells and nets in the transitive fanout of **dont_touch_network** ports. The **dont_touch** assignment stops at the boundary of storage elements. An element is recognized as storage only if it has setup or hold constraints. Set with **set_dont_touch_network.**

fall_delay

Specifies an offset from the falling edge of the ideal clock waveform. Set with **set_clock_skew -fall_delay**.

fix_hold

Specifies that **compile** should attempt to fix hold violations for timing endpoints related to this clock. Set with **set_fix_hold**.

max_fall_delay

A floating point value that specifies the maximum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

max_rise_delay

A floating point value that specifies the maximum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

max_time_borrow

A floating point number that establishes an upper limit for time borrowing; that is, it prevents the use of the entire pulse width for level-sensitive latches. Units are those used in the technology library. Set with **set_max_time_borrow**.

min_fall_delay

A floating point value that specifies the minimum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

min_rise_delay

A floating point value that specifies the minimum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

minus_uncertainty

Specifies a negative uncertainty from the edges of the ideal clock waveform. Set with **set_clock_skew -minus_uncertainty**.

period

Assigns a value to the clock period. The clock period (or cycle time) is the shortest time during which the clock waveform repeats. For a simple waveform with one rising and one falling edge, the period is the difference between successive rising edges. Set with **create_clock -period_value**.

plus_uncertainty

Specifies a positive uncertainty from the edges of the ideal clock waveform. Affects all sequential cells in the transitive fanout of this port. Set with **set_clock_skew -plus_uncertainty**.

propagated_clock
Specifies that the clock edge times be delayed by propagating the values through the clock network. If this attribute is not present, ideal clocking is assumed. Set with **set_clock_skew -propagated**.

rise_delay
Specifies an offset from the rising edge of the ideal clock waveform. Set with **set_clock_skew -rise_delay**

Design Attributes

async_set_reset_q
Establishes the value (0 or 1) that should be assigned to the q output of an inferred register if set and reset are both active at the same time. To be used with **async_set_reset_qn**. Use these attributes only if you have used the **one_hot** or **one_cold** attributes/directives in your HDL description *and* your technology library is written using pre-V3.0a syntax; *or* if your technology library does not use a consistent convention for q and qn when set and reset are both active. If a V3.0a or later syntax technology library is used, then by default if set and reset are both active at the same time Design Compiler will use the convention of the selected technology library (**target_library**). Set with **set_attribute**.
Note: If you are unsure whether or not your technology library uses V3.0a syntax, ask your ASIC vendor.

async_set_reset_qn
Establishes the value (0 or 1) that should be assigned to the qn output of an inferred register if set and reset are both active at the same time. To be used with **async_set_reset_q**. Use these attributes only if you have used the **one_hot** or **one_cold** attributes/directives in your HDL description *and* your technology library is written using pre-V3.0a syntax; *or* if your technology library does not use a consistent convention for q and qn when set and reset are both active. If a V3.0a or later syntax technology library is used, then by default if set and reset are both active at the same time Design Compiler will use the convention of the selected technology library (**target_library**). Set with **set_attribute**.
Note: If you are unsure whether or not your technology library uses V3.0a syntax, ask your ASIC vendor.

balance_registers
Determines whether the registers in a design are retimed during **compile**. When *true* (the default value), **compile** invokes the **balance_registers** command, which moves registers to minimize the maximum register-to-register delay. Set this attribute to *false*, or remove it, to disable this behavior.
Set with **set_balance_registers**. **Note:** You cannot verify designs using **compile -verify** while the **balance_registers** attribute is set to *true*. In addition, if your design contains generic logic, you should ensure that all components are mapped to cells from the library before setting the **balance_registers** attribute.

boundary_optimization
Enables **compile** to optimize across hierarchical boundaries. Hierarchy is ignored during optimization for designs with this attribute set to *true*. Set with **set_boundary_optimization** .

default_flip_flop_type

Specifies the default flip-flop type for the current design. During the mapping process, **compile** tries to convert all unmapped flip-flops to this type. If **compile** is unable to use this flip-flop, it maps these cells into the smallest flip-flop possible. Set with **set_register_type -flip_flop** *flip_flop_name*.

default_flip_flop_type_exact

During the mapping process, **compile** converts unmapped flip-flops to the exact flip-flop type specified here. Set with **set_register_type -exact -flip_flop** *flip_flop_name*

default_latch_type_exact

Specifies the exact default latch type for the **current_design**. During the mapping process, **compile** converts unmapped latches to the exact latch type specified here. Set with **set_register_type -exact -latch** *latch_name*.

design_type

Indicates the current state of the design and has the value *fsm* (finite state machine), *pla* (programmable logic array), *equation* (Boolean logic), or *netlist* (gates). This attribute is "read-only" and cannot be set by the user.

dont_touch

Identifies designs that are to be excluded from optimization. Values are *true* (the default) or *false*. Designs with the **dont_touch** attribute set to *true* are not modified or replaced during **compile**. Setting **dont_touch** on a design has an effect only when the design is instantiated within another design as a level of hierarchy; setting **dont_touch** on the top-level design has no effect. Set with **set_dont_touch** .

flatten

When set to *true*, determines that a design is to be flattened during **compile**. By default, a design is not flattened. Set with **set_flatten**.

flatten_effort

Defines the level of CPU effort that **compile** uses to flatten a design. Allowed values are *low* (the default), *medium*, or *high*. Set with **set_flatten** .

flatten_minimize

Defines the minimization strategy used for logic equations. Allowed values are *single_output*, *multiple_output*, or *none*. Set with **set_flatten** .

flatten_phase

When *true*, allows logic flattening to invert the phase of outputs during **compile**. By default, logic flattening does not invert the phase of outputs. Used only if the **flatten** attribute is set. Set with **set_flatten** .

implementation

The implementation for each specified instance of the specified component_type. Specifying the **-default** option removes this attribute from all instances of the component type in the current design. Set with **set_jtag_implementation**.

is_combinational

true if all cells of a design and all designs in its hierarchy are combinational. A cell is combinational if it is non-sequential or nonthree-state and all of its outputs compute a combinational logic function. The **report_lib** command will report such a cell as not a black-box. This attribute is *read-only*; you cannot set it.

is_dw_subblock
: *true* if the object (a cell, a reference, or a design) is a DW subblock that was automatically elaborated. This attribute is "read-only" and cannot be set by the user.
NOTE: DW subblocks that are manually elaborated will not have this attribute.

is_hierarchical
: *true* if any of the cells of a design are not leaf cells (for example, not from a technology library). This attribute is *read-only* and cannot be set by the user.

is_mapped
: *true* if all the non-hierarchical cells of a design are mapped to cells in a technology library. This attribute is "read-only" and cannot be set by the user.

is_sequential
: *true* if any cells of a design or designs in its hierarchy are sequential. A cell is sequential if it is not combinational (if any of its outputs depend on previous inputs). This attribute is *read-only* and cannot be set by the user.

is_synlib_module
: *true* if the object (a cell, a reference, or a design) refers to an unmapped module reference or if the object is (or refers to) a design that was automatically elaborated from a synlib module or a synlib operator. This attribute is "read-only" and cannot be set by the user.
NOTE: synlib modules that are manually elaborated will not have this attribute.

is_unmapped
: *true* if any of the cells are not linked to a design or mapped to a technology library. This attribute is *read-only* and cannot be set by the user.

local_link_library
: A string that contains a list of design files and libraries to be added to the beginning of the **link_library** whenever a **link** operation is performed. Set with **set_local_link_library**.

map_only
: When set to *true*, **compile** will attempt to map the object exactly in the target library, and will exclude the object from logic-level optimization (flattening and structuring). The default is *false*. Set with **set_map_only**.

max_area
: A floating point number that represents the target area of the design. **compile** uses it to calculate the area cost of the design. The units must be consistent with the units used from the technology library during optimization. Set with **set_max_area**.

max_capacitance

A floating point number that sets the maximum capacitance value for input, output, or bidirectional ports, or designs. The units must be consistent with those of the technology library used during optimization. Set with **set_max_capacitance**.

max_dynamic_power

A floating point number that specifies the maximum target dynamic power for the **current_design**. The units must be consistent with those of the technology library. If this attribute is specified more than once for a design, the latest value is used. Set with **set_max_dynamic_power**.

max_leakage_power

A floating point number that specifies the maximum target leakage power for the **current_design**. The units must be consistent with those of the technology library. If this attribute is specified more than once for a design, the latest value is used. Set with **set_max_leakage_power**.

max_total_power

A floating point number that specifies the maximum target total power for the **current_design**. Total power is defined as the sum of dynamic and leakage power. If this attribute is specified more than once for a design, the latest value is used. Set with **set_max_total_power**.

min_porosity

Specifies the minimum porosity of the design. **compile -routability** and **reoptimize_design** ensure that the porosity of the design is greater than the specified value. Set with **set_min_porosity** .

minimize_tree_delay

When *true* (the default value), **compile** restructures expression trees in the **current_design** or in a list of specified designs, to minimize tree delay. The value of this attribute overrides the value of **hlo_minimize_tree_delay**. Set this attribute to *false* for any designs that you do not wish to be restructured. Set with **set_minimize_tree_delay**.

model_map_effort

Specifies the relative amount of CPU time to be used by **compile** during modeling, typically for synthetic library implementations. Values are *low*, *medium*, and *high*, or *1*, *2*, and *3*. If **model_map_effort** is not set, the value of **synlib_model_map_effort** is used. Set with **set_model_map_effort**.

model_scale

A floating point number that sets the model scale factor for the **current_design**. Set with **set_model_scale**.

optimize_registers

When *true* (the default value), **compile** automatically invokes the Behavioral Compiler **optimize_registers** command to retime the design during optimization. Setting the attribute to *false* disables this behavior. You cannot execute **compile -verify** if **optimize_registers** is set to *true* on the design; and your design cannot contain generic logic at the instant **optimize_registers** is invoked during **compile**. Set with **set_optimize_registers**.

part

A string value that specifies the Xilinx part type for a design. For valid part types, refer to the Xilinx *XC4000 Databook*. Set with **set_attribute**.

port_is_pad

Indicates specified ports are to have I/O pads attached. The I/O pads are added during insert_pads and automatically added during compile. Set using **set_port_is_pad**.

resource_allocation

Indicates the type of resource allocation to be used by **compile** for the **current_design**. Allowed values are *none*, indicating no resource sharing; *area_only*, indicating resource sharing with tree balancing without considering timing constraints; *area_no_tree_balancing*, indicating resource sharing without tree balancing and without considering timing constraints; and *constraint_driven* (the default), indicating resource sharing so that timing constraints are met or not worsened. The value of this attribute overrides the value of the variable **hlo_resource_allocation** for the **current_design**. Set with **set_resource_allocation**.

resource_implementation

Indicates the type of resource implementation to be used by **compile** for the **current_design**. Allowed values are *area_only*, indicating resource implementation without considering timing constraints; *constraint_driven*, indicating resource implementation so that timing constraints are met or not worsened; and *use_fastest*, indicating resource implementation using the fastest implementation initially, unless all timing constraints are met. If the fastest implementation has been selected initially later steps of the compile command will select components with smaller area later in uncritical parts of the design. The value of this attribute overrides the value of the variable **hlo_resource_implementation** for the **current_design**. Set with **set_resource_implementation**.

scan_element

Determines if sequential cells in the specified designs are replaced by equivalent scan cells or designs during **insert_scan**. Default is set to *true*. When set to *false*, sequential cels are not replaced by equivalent scan cells. Set using **set_scan_element**.

scan_latch_transparent

When set to *true*, makes specified designs transparent during ATPG. For hierarchical cells, the effects apply hierarchically to level-sensitive leaf cells. The **set_scan_transparent** command sets the attribute; the **remove_attribute** command removes it.

share_cse

When *true*, the value of the environment variable **hlo_share_common_subexpressions** is used. The value of this attribute determines whether common subexpressions are shared during compile, to reduce the cost of the design. Setting the attribute to *false* overrides the **hlo_share_common_subexpressions**. Set with **set_share_cse**.

structure

Determines if a design is to be structured during **compile**. If *true*, adds logic structure to a design by adding intermediate variables that are factored out of the design’s equations. Set with **set_structure** .

structure_boolean
Enables the use of Boolean (non-algebraic) techniques during the structuring phase of optimization. This attribute is ignored if the **structure** attribute is *false* Set with **set_structure** .

structure_timing
Enables timing constraints to be considered during the structuring phase of optimization. This attribute is ignored if the **structure** attribute is *false*. Set with **set_structure** .

ungroup
Removes a level of hierarchy from the current design by exploding the contents of the specified cell in the current design. Set with **set_ungroup** .

wired_logic_disable
When *true*, disables creation of wired OR logic during **compile**. The default is *false*; if this attribute is not set, wired OR logic will be created if appropriate. Set with **set_wired_logic_disable**.

wire_load_model_mode
Determines which wire load model to use to compute wire capacitance, resistance, and area for nets in a hierarchical design that has different wire load models at different hierarchical levels. Allowed values are *top*, which indicates to use the wire load model at the top hierarchical level; *enclosed*, which indicates to use the wire load model on the smallest design that encloses a net completely; and *segmented*, which indicates to break the net into segments, one within each hierarchical level. In the *segmented* mode, each net segment is estimated using the wire load model on the design that encloses that segment. The *segmented* mode is not supported for wire load models on clusters. If a value is not specified for this attribute, **compile** searches for a default in the first library in the link path. If none is found, *top* is the default. Set with **set_wire_load**.

xnfout_use_blknames
When *true*, the Synopsys XNF writer writes BLKNM XNF parameters into the XNF netlist for your design when **write -f xnf** is invoked. The default is *false*. The BLKNM XNF parameters convey to the Xilinx place and route tools information, previously placed on the **db_design** by **replace_fpga**, that indicates which groupings of function generators are to be packed into CLB cells. Set with **set_attribute**.

Library Cell Attributes

dont_touch
Identifies library cells to be excluded from optimization. Values are *true* (the default) or *false*. Library cells with the **dont_touch** attribute set to *true* are not modified or replaced during **compile**. Setting **dont_touch** on a hierarchical cell sets the attribute on all cells below it. Set with **set_dont_touch** .

dont_use
Disables the specified library cells so that they are not added to a design

during **compile**. Set with **set_dont_use** .

formula

The attribute of the priority parameter for implementations in synthetic libraries. The formula should evaluate to an integer between 0 and 10. Set with **set_impl_priority**.

implementation

Specifies the implementation for the specifid synthetic library cell instances to use. When compile is run, the implementation you specified is used if you set this attribute. The cells instances must be defined in the synthetic library for this attribute to work. Set with **set_implementation**.

no_sequential_degenerates

When *true*, disables mapping to versions of this latch or flip flop that have some input pins connected to 0 or to 1. Set with **set_attribute**. This attribute may also be set on the library itself, and that value will apply as the default for all registers in the library which don't have the attribute set individually.

preferred

Specifies the preferred library gate to use during technology translation when there are other gates with the same function in the target library. Set with **set_prefer** .

scan

When *true*, specifies that the instances of the library cell are always replaced by equivalent scan cells during **insert_dft**. When false, instances are not replaced. Set with **set_scan.**

scan_group

A user-defined string variable that allows you to specify to DFT Compiler a preferred scan equivalent for a non-scan storage element, when a library contains multiple scan equivalents. Typical values are *low*, *medium*, and *high*, for low, medium, and high drive strengths. However, you can define any string variable, and it need not describe drive strength. The default behavior is for DFT Compiler to attempt to choose a scan element that best matches the electrical characteristics of the nonscan element; for a more detailed explanation, refer to the *DFT Compiler Scan Synthesis User Guide*. The matching of electrical characteristics works well with the standard CMOS delay model, but is not accurate with other delay models; **scan_group** provides a means for you to specify an appropriate scan equivalent. Normally, **scan_group** would be set by the ASIC vendor or library developer, but can also be set by you. Consult your ASIC vendor before attempting to set **scan_group** with **set_attribute**. For more information about **scan_group**, refer to the *DFT Compiler Scan Synthesis User Guide*.

set_id

Allows for the value for the implementations in synthetic libraries. Set with **set_impl_priorities**.
Determines if specified designs are scan replaced by **insert_scan**. Set using **set_scan_element**.

scan_latch_transparent

When *true*, makes the specified library cells transparent in ATPG. For hierarchical cells, the effects apply hierarchically to level-sensitive leaf cells. The **set_scan_transparent** command sets the attribute; the **remove_attribute** command removes it.

sequential_bridging

When *true*, enables **Design Compiler** to take a multiplexed flip-flop and bridge (that is, connect) the output to the input to get a desired functionality. The default is *false*, so this attribute must be set in order to enable the functionality. Bridging is required for mapping in cases where there is no flip-flop with internal feedback in the target library but one is desired in the HDL. Set with **set_attribute**. This attribute may also be set on the library itself, and that value will apply as the default for all registers in the library which don't have the attribute set individually.
NOTE: Setting this attribute to *true* can result in an increase in run times and memory consumption for Design Compiler. The increased run times depend on the number of flip-flops in the target library or libraries for which this attribute has been set.

Net Attributes

actual_max_net_capacitance

actual_min_net_capacitance

A floating point number that specified the total calculated capacitance of the net. The value of these attributes is calculated upon request. These are "read-only" attributes and they cannot be set by the user.

ba_net_resistance

A floating point number that specifies the back-annotated net resistance on a net. Set with **set_resistance**.

dont_touch

Identifies nets to be excluded from optimization. Values are *true* (the default) or *false*. Nets with the **dont_touch** attribute set to *true* are not modified or replaced during **compile**. Set with **set_dont_touch** .

is_test_circuitry

Set by **insert_dft** on the scan cells and nets added to a design during the addition of test circuitry. This attribute is "read-only" and cannot be set by the user.

"load *"

A floating point number that specifies the wire load value on a net. The total load on a net is the sum of all the loads on pins, ports, and wires associated with that net. This attribute represents only the wire load; pin and port loads are not included in the attribute value unless the attribute *subtract_pin_load* is set to **true** for the same net. Set with **set_load.**

static_probability

A floating point number that specifies the percentage of time that the signal is in the logic 1 state; this information is used by **report_power**. If this attribute is not set, **report_power** will use the default value of 0.5, indicating that the signal is in the logic 1 state half the time. Set with **set_switching_activity**.

subtract_pin_load

Causes **compile** to reduce the wire load value of a net by an amount equal to its pin load. Specifies that the **load** attribute includes the capacitances of all pins on the net. If the resulting wire load is negative, it is set to zero. Set with **set_load -subtract_pin_load**.

toggle_rate

A positive floating point number that specifies the toggle rate; that is, the number of zero-to-one and one-to-zero transitions within a library time unit period. This information is used by **report_power**; if this attribute is not set, **report_power** will use the default value of 2*(static_probability)(1 - static_probability). The default will be scaled by any associated clock signal (if one is available). Set with **set_switching_activity**.

wired_and

One of a set of two wired logic attributes that includes **wired_or**. When present and set to *true*, **wired_and** determines that the associated net has more than one driver and implements a wired AND function. Wired logic attributes cannot be manually set by the user. To cause wired logic attributes to be added to a netlist design that contains multiply-driven nets, you have two alternatives: 1. execute **compile** or **translate** on the design; or 2. specify the wired logic types using a resolution function in the HDL file.

wired_or

One of a set of two wired logic attributes that includes **wired_and**. When present and set to *true*, **wired_or** determines that the associated net has more than one driver and implements a wired OR function. Wired logic attributes cannot be manually set by the user. To cause wired logic attributes to be added to a netlist design that contains multiply-driven nets, you have two alternatives: 1. execute **compile** or **translate** on the design; or 2. specify the wired logic types using a resolution function in the HDL file.

Pin Attributes

actual_max_net_capacitance

actual_min_net_capacitance

A floating point number that specified the total calculated capacitance of the net that is connected to the given pin. The attributes are defined only for pins of leaf cell. The value of these attributes is calculated upon request. These are "read-only" attributes and they cannot be set by the user.

disable_timing

Disables timing arcs. This has the same effect on timing as not having the arc in the library. Set with **set_disable_timing.**

fall_delay

Specifies an offset from the falling edge of the ideal clock waveform. Set with **set_clock_skew -fall_delay**.

max_slack

A floating point value representing the worst slack of **max_rise_slack** and **max_fall_slack**.

max_fall_slack
A floating point value representing the worst slack at a pin for falling maximum path delays. This attribute is valid for any pin that appears in a constrained path after timing has been updated.

max_rise_slack
A floating point value representing the worst slack at a pin for rising maximum path delays. This attribute is valid for any pin that appears in a constrained path after timing has been updated.

min_slack
A floating point value representing the worst slack of **min_rise_slack** and **min_fall_slack**.

min_fall_slack
A floating point value representing the worst slack at a pin for falling minimum path delays. This attribute is valid for any pin that appears in a constrained path after timing has been updated..

min_rise_slack
A floating point value representing the worst slack at a pin for rising minumum path delays. This attribute is valid for any pin that appears in a constrained path after timing has been updated.

max_fall_delay
A floating point value that specifies the maximum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

max_rise_delay
A floating point value that specifies the maximum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

min_fall_delay
A floating point value that specifies the minimum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

min_rise_delay
A floating point value that specifies the minimum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

minus_uncertainty
Specifies a negative uncertainty from the edges of the ideal clock waveform. Set with **set_clock_skew -minus_uncertainty**.

observe_pin
Specifies the (internal) observe pin name of an LSI Logic scan macrocell (LSI CTV only). This attribute is used by the **write_test** command. Set with **set_attribute** .

pin_direction
Specifies the direction of a pin. Allowed values are *in*, *out*, *inout*, or

unknown. This attribute is **read-only** and cannot be set by the user.

pin_properties
: Lists valid EDIF property values to be attached to different versions of the output pin. The EDIF property values correspond to different output emitter-follower resistance values on the output pin. For details about the use of this attribute, refer to the *Library Compiler Reference Manual*, Chapter 6, "Defining Cells." Set with **set_attribute**.

plus_uncertainty
: Specifies a positive uncertainty from the edges of the ideal clock waveform. Affects all sequential cells in the transitive fanout of this port. Set with **set_clock_skew -plus_uncertainty**.

propagated_clock
: Specifies that the clock edge times be delayed by propagating the values through the clock network. If this attribute is not present, ideal clocking is assumed. Set with **set_clock_skew -propagated**.

rise_delay
: Specifies an offset from the rising edge of the ideal clock waveform. Set with **set_clock_skew -rise_delay**.

set_pin
: Specifies the (internal) set pin name of an LSI Logic scan macrocell (LSI CTV only). This attribute is used by the **write_test** command. Set with **set_attribute** .

signal_type
: Used to indicate that a pin or port is of a special type, such as a **clocked_on_also** port in a master/slave clocking scheme, or a **test_scan_in** pin for scan-test circuitry. Set with **set_signal_type** .

static_probability
: A floating point number that specifies the percentage of time that the signal is in the logic 1 state; this information is used by **report_power**. If this attribute is not set, **report_power** will use the default value of 0.5, indicating that the signal is in the logic 1 state half the time. Set with **set_switching_activity**.

test_assume
: A string that represents a constant logic value to be assumed for specified pins throughout test design rule checking by **check_test**. "1", "one", or "ONE" specifies a constant value of logic one; "0", "zero", or "ZERO" specifies a constant value of logic zero. Use **report_test -assertions** for a report on objects that have the **test_assume** attribute set. Set with **set_test_assume** .

test_initial
: A string that represents an initial logic value to be assumed for specified pins at the start of test design rule checking and fault simulation by **check_test**. "1", "one", or "ONE" specifies an initial value of logic one; "0", "zero", or "ZERO" specifies an initial value of logic zero. Use **report_test -assertions** for a report on objects that have the **test_initial** attribute set. Set with **set_test_initial**.

test_isolate

Indicates that the specified sequential cells, pins, or ports are to be logically isolated and considered untestable during test design rule checking by **check_test**. When this attribute is set on a cell, it is also placed on all pins of that cell. Do not set this attribute on a hierarchical cell. Use **report_test -assertions** for a report on isolated objects. Set with **set_test_isolate**.
Note: Setting this attribute suppresses the warning messages associated with the isolated objects.

test_routing_position

Specifies the preferred routing order of the scan-test signals of the identified cells. Set with **set_test_routing_order** .

toggle_rate

A positive floating point number that specifies the toggle rate; that is, the number of zero-to-one and one-to-zero transitions within a library time unit period. This information is used by **report_power**; if this attribute is not set, **report_power** will use the default value of 2*(static_probability)(1 - static_probability). The default will be scaled by any associated clock signal (if one is available). Set with **set_switching_activity**.

true_delay_case_analysis

Specifies a value to set all or part of an input vector for **report_timing -true** and **report_timing -justify**. Allowed values are *0*, *1*, *r* (rise, X to 1), and *f* (fall, X to 0). Set with **set_true_delay_case_analysis.**

Port Attributes

actual_max_net_capacitance

actual_min_net_capacitance

A floating point number that specified the total calculated capacitance of the net connected to the given port. The value of these attributes is calculated upon request. These are "read-only" attributes and they cannot be set by the user.

connection_class

A string that specifies the connection class label to be attached to a port or to a list of ports. **compile**, **insert_pads**, and **insert_dft** will connect only those loads and drivers that have the same connection class label. The labels must match those in the library of components for the design, and must be separated by a space. The labels *universal* and *default* are reserved; *universal* indicates that the port can connect with any other load or driver, and *default* is assigned to any ports that do not have a connection class already assigned. Set with **set_connection_class**.

dont_touch_network

When a design is optimized, **compile** assigns **dont_touch** attributes to all cells and nets in the transitive fanout of **dont_touch_network** clock objects. The **dont_touch** assignment stops at the boundary of storage elements. An element is recognized as storage only if it has setup or hold constraints. Set with **set_dont_touch_network** .

driven_by_dont_care

Specifies that input port are driven by dont_care. Compile uses this information to create smaller designs. After optimization, the port connected to dont_care does not drive anything inside the optimized design. Set with **set_logic_dc**.

driven_by_logic_one

Specifies that input ports are driven by logic one. **compile** uses this information to create smaller designs. After optimization, a port connected to logic one usually does not drive anything inside the optimized design. Set with **set_logic_one** .

driven_by_logic_zero

Specifies that input ports are driven by logic zero. **compile** uses this information to create smaller designs. After optimization, a port connected to logic zero usually does not drive anything inside the optimized design. Set with **set_logic_zero** .

driving_cell_dont_scale

When *true*, indicates not to scale the transition time on the port using the driving cell. Otherwise the transition time will be scaled by operating condition factors. Set with **set_driving_cell**.

driving_cell_fall

A string that names a library cell from which to copy fall drive capability to be used in fall transition calculation for the port. Set with **set_driving_cell**.

driving_cell_from_pin_fall

A string that names the driving_cell_fall input pin to be used to find timing arc fall drive capability. Set with **set_driving_cell**.

driving_cell_from_pin_rise

A string that names the driving_cell_rise input pin to be used to find timing arc rise drive capability. Set with **set_driving_cell**.

driving_cell_library_fall

A string that names the library in which to find the **driving_cell_fall**. Set with **set_driving_cell**.

driving_cell_library_rise

A string that names the library in which to find the **driving_cell_rise**. Set with **set_driving_cell**.

driving_cell_multiply_by

A floating point value by which to multiply the transition time of the port marked with this attribute. Set with **set_driving_cell**.

driving_cell_pin_fall

A string that names the driving_cell_fall output pin to be used to find timing arc fall drive capability. Set with **set_driving_cell**.

driving_cell_pin_rise

A string that names the driving_cell_rise output pin to be used to find timing arc rise drive capability. Set with **set_driving_cell**.

driving_cell_rise
: A string that names a library cell from which to copy rise drive capability to be used in rise transition calculation for the port. Set with **set_driving_cell**.

fall_delay
: Specifies an offset from the falling edge of the ideal clock waveform. Set with **set_clock_skew -fall_delay**

fall_drive
: Specifies the drive value of high to low transition on input or inout ports. Set with **set_drive.**

fanout_load
: Specifies the fanout load on output ports. Set with **set_fanout_load** .

load
: Specifies the load value on ports. The total load on a net is the sum of all the loads on pins, ports, and wires associated with that net. Set with **set_load.**

max_capacitance
: A floating point number that sets the maximum capacitance value for input, output, or bidirectional ports, and/or designs. The units must be consistent with those of the technology library used during optimization. Set with **set_max_capacitance**.

max_fall_delay
: A floating point value that specifies the maximum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

max_fanout
: Specifies the maximum fanout load for the net connected to this port. **compile** ensures that the fanout load on this net is less than the specified value. Set with **set_max_fanout** .

max_rise_delay
: A floating point value that specifies the maximum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

max_slack
: A floating point value representing the worst slack of **max_rise_slack** and **max_fall_slack**.
.XA max_fall_slack A floating point value representing the worst slack at a pin for falling maximum path delays. This attribute is valid for any pin that appears in a constrained path after timing has been updated.
.XA max_rise_slack A floating point value representing the worst slack at a pin for rising maximum path delays. This attribute is valid for any pin that appears in a constrained path after timing has been updated.

min_slack
: A floating point value representing the worst slack of **min_rise_slack** and **min_fall_slack**.

.XA min_fall_slack A floating point value representing the worst slack at a pin for falling minimum path delays. This attribute is valid for any pin that appears in a constrained path after timing has been updated.
.XA min_rise_slack A floating point value representing the worst slack at a pin for rising minumum path delays. This attribute is valid for any pin that appears in a constrained path after timing has been updated.

max_time_borrow
A floating point number that establishes an upper limit for time borrowing; that is, it prevents the use of the entire pulse width for level-sensitive latches. Units are those used in the technology library. Set with **set_max_time_borrow**.

max_transition
Specifies the maximum transition time for the net connected to this port. **compile** ensures that value. Set with **set_max_transition** .

min_capacitance
A floating point number that sets the minimum capacitance value for input and/or bidirectional ports. The units must be consistent with those of the technology library used during optimization. Set with **set_min_capacitance**.

min_fall_delay
A floating point value that specifies the minimum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

min_rise_delay
A floating point value that specifies the minimum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

minus_uncertainty
Specifies a negative uncertainty from the edges of the ideal clock waveform. Affects all sequential cells in the transitive fanout of this port. Set with **set_clock_skew -minus_uncertainty**.

model_drive
A non-negative floating point number that specifies the estimated drive value on ports in terms of standard drives of the current technology library. Set with **set_model_drive**.

model_load
A non-negative floating point number that specifies the estimated load value on ports in terms of standard loads of the current technology library. Set with **set_model_load**.

op_used_in_normal_op
Specifies that a scan-out port is also used in normal operation (system mode). This attribute is used by the **insert_dft** command. Set with **set_attribute**.

output_not_used
Determines that an output port is unconnected. Used by **compile** to create smaller designs since the logic that drives an unconnected output port might not need to be maintained. After a design with an unconnected output port is

compiled, the port is usually not driven by anything inside the design. Set with **set_unconnected** .

pad_location
: A string value that specifies the Xilinx pad location (pin number) to be assigned to a port. Setting a **pad_location** attribute on a port causes the Synopsys XNF writer to indicate in the XNF netlist that this port has the pad location given by the value of the **pad_location** attribute. No checks are performed to verify that the specified location is valid; for valid pad locations, refer to the Xilinx *XC4000 Databook*. Set with **set_attribute**.

plus_uncertainty
: Specifies a positive uncertainty from the edges of the ideal clock waveform. Affects all sequential cells in the transitive fanout of this port. Set with **set_clock_skew -plus_uncertainty**.

port_direction
: Direction of a port. Value can be *in*, *out*, *inout*, or *unknown*. This attribute is "read-only" and cannot be set by the user.

port_is_pad
: Indicates specified ports are to have I/O pads attached. The I/O pads are added using **insert_pads** and automatically optimized during compile. Set using **set_port_is_pad**.

propagated_clock
: Specifies that the clock edge times be delayed by propagating the values through the clock network. Affects all sequential cells in the transitive fanout of this port. If this attribute is not present, ideal clocking is assumed. Set with **set_clock_skew -propagated**.

rise_delay
: Specifies an offset from the rising edge of the ideal clock waveform. Set with **set_clock_skew -rise_delay**.

rise_drive
: Specifies the drive value of low to high transition on input or inout ports. Set with **set_drive.**

signal_index
: Used to enumerate different ports with the same signal type (for example, scan-in ports for a design with multiple scan chains). Set with **set_signal_type** .

signal_type
: Used to indicate that a port is of a "special" type, such as a "clocked_on_also" port in a master/slave clocking scheme, or a "test_scan_in" pin for scan-test circuitry. Set with **set_signal_type** .

static_probability
: A floating point number that specifies the percentage of time that the signal is in the logic 1 state; this information is used by **report_power**. If this attribute is not set, **report_power** will use the default value of 0.5, indicating that the signal is in the logic 1 state half the time. Set with **set_switching_activity**.

test_hold

Specifies the fixed, constant logic value at a port during test generation. Set with **set_test_hold** .

test_isolate

Indicates that the specified sequential cells, pins, or ports are to be logically isolated and considered untestable during test design rule checking by **check_test**. When this attribute is set on a cell, it is also placed on all pins of that cell. Do not set this attribute on a hierarchical cell. Use **report_test -assertions** for a report on isolated objects. Set with **set_test_isolate**.
Note: Setting this attribute suppresses the warning messages associated with the isolated objects.

toggle_rate

A positive floating point number that specifies the toggle rate; that is, the number of zero-to-one and one-to-zero transitions within a library time unit period. This information is used by **report_power**; if this attribute is not set, **report_power** will use the default value of 2*(static_probability)(1 - static_probability). The default will be scaled by any associated clock signal (if one is available). Set with **set_switching_activity**.

true_delay_case_analysis

Specifies a value to set all or part of an input vector for **report_timing -true** and **report_timing -justify**. Allowed values are *0*, *1*, *r* (rise, X to 1), and *f* (fall, X to 0). Set with **set_true_delay_case_analysis.**

Read-Only Attributes

design_type

Indicates the current state of the design and has the value *fsm* (finite state machine), *pla* (programmable logic array), *equation* (Boolean logic), or *netlist* (gates). This attribute cannot be set by the user.

is_black_box

true if the reference is not yet linked to a design or is linked to a design that doesn’t have a functionality. This attribute cannot be set by the user.

is_combinational

true if all cells of a design and all designs in its hierarchy are combinational. A cell is combinational if it is non-sequential or non-tristate and all of its outputs compute a combinational logic function. The **report_lib** command will report such a cell as not a black-box. This attribute is *read-only* and cannot be set by the user.

is_dw_subblock

true if the object (a cell, a reference, or a design) is a DW subblock that was automatically elaborated. This attribute is "read-only" and cannot be set by the user.

is_hierarchical

true if any of the cells of a design are not leaf cells (for example, not from a technology library). This attribute cannot be set by the user.

is_mapped

true if all the non-hierarchical cells of a design are mapped to cells in a technology library. This attribute cannot be set by the user.

is_sequential

true if any cells of a design or designs in its hierarchy are sequential. A cell is sequential if it is not combinational. This attribute cannot be set by the user.

is_synlib_module

true if the object (a cell, a reference, or a design) refers to an unmapped module reference or if the object is (or refers to) a design that was automatically elaborated from a synlib module or a synlib operator. This attribute is "read-only" and cannot be set by the user.
NOTE: synlib modules that are manually elaborated will not have this attribute.

is_synlib_operator

true if the object (a cell or a reference) is a synthetic library operator reference. This attribute is "read-only" and cannot be set by the user.

is_test_circuitry

Set by **insert_dft** on the scan cells and nets added to a design during the addition of test circuitry. This attribute cannot be set by the user.

is_unmapped

true if any of the cells are not linked to a design or mapped to a technology library. This attribute cannot be set by the user.

pin_direction

Direction of a pin. Value can be *in*, *out*, *inout*, or *unknown*. This attribute cannot be set by the user.

port_direction

Direction of a port. Value can be *in*, *out*, *inout*, or *unknown*. This attribute cannot be set by the user.

ref_name

The reference name of a cell. This attribute cannot be set by the user.

Reference Attributes

dont_touch

Specifies that designs linked to a reference with this attribute are excluded from optimization. Values are *true* (the default) or *false*. Designs linked to a reference with with the **dont_touch** attribute set to *true* are not modified or replaced during **compile**. Set with **set_dont_touch** .

is_black_box

true if the reference is not yet linked to a design or is linked to a design that doesn't have a functionality. This attribute is *read-only* and cannot be set by the user.

is_combinational

true if all the cells of the referenced design are combinational. A cell is combinational if it is non-sequential or non-tristate and all of its outputs compute a combinational logic function. The **report_lib** command will report such a cell as not a black-box. This attribute is *read-only* and cannot be set by the user.

is_dw_subblock

true if the object (a cell, a reference, or a design) is a DW subblock that was automatically elaborated. This attribute is *read-only* and cannot be set by the user.
NOTE: DW subblocks that are manually elaborated will not have this attribute.

is_hierarchical

true if the referenced design is not a leaf cell (for example, not in a technology library). This attribute is *read-only* and cannot be set by the user.

is_mapped

true if the reference is linked to a design, and all the non-hierarchical cells of the referenced design are mapped to cells in a technology library. This attribute is *read-only* and cannot be set by the user.

is_sequential

true if all the cells of the referenced design are sequential. A cell is sequential if it is not combinational (if any of its outputs depend on previous inputs). This attribute is *read-only* and cannot be set by the user.

is_synlib_module

true if the object (a cell, a reference, or a design) refers to an unmapped module reference or if the object is (or refers to) a design that was automatically elaborated from a synlib module or a synlib operator. This attribute is *read-only* and cannot be set by the user.
NOTE: synlib modules that are manually elaborated will not have this attribute.

is_synlib_operator

true if the object (a cell or a reference) is a synthetic library operator reference. This attribute is *read-only* and cannot be set by the user.

is_unmapped

true if any of the non-hierarchical cells of the referenced design are not mapped to cells in a technology library, or if the reference is not yet linked to a design. This attribute is *read-only* and cannot be set by the user.

scan

When *true*, specifies that cells of the referenced design are always replaced by equivalent scan cells during **insert_dft**. When false, cells of the design are not replaced. Set with **set_scan** .

scan_chain

Includes the specified cells of the referenced design in the scan-chain whose index is the value of this attribute. Set with **set_scan_chain** .

scan_element
    Determines if specified designs are scan replaced by **insert_scan**. Set using **set_scan_element**.

scan_latch_transparent
    When *true*, makes the specified references transparent in ATPG. For hierarchical cells, the effects apply hierarchically to level-sensitive leaf cells. The specified library cell cannot be overwritten. Set with **set_scan_transparent**; remove with **remove_attribute**.

ungroup
    Specifies that all designs linked to a reference with this attribute are ungrouped (levels of hierarchy represented by these design cells are removed) during **compile**. Set with **set_ungroup** .

utilization
    Specifies the utilization of a soft macro.
    We calculate the utilization using **physical_area**:**area** ratio of a soft macro.
    This attribute is "read-only" and cannot be set by the user.

width
    Specifies the width of a cell. This attribute doesn't exist on a hierarchical cell.
    We use the cell's cell boundary to calculate its width.
    This attribute is "read-only" and cannot be set by the user.

SEE ALSO

find (2), **get_attribute** (2), **remove_attribute** (2), **set_attribute** (2), **power_attributes** (3).

auto_insert_level_shifters

Setting this variable to false would prevent automatic level shifter insertion in commands like compile, insert_dft etc

TYPE

Boolean

DEFAULT

true

GROUP

mv

DESCRIPTION

By default this variable is true. This enables commands like compile to automatically insert level shifters where needed. Setting it to false disable this feature.

auto_insert_level_shifters_on_clocks

This variable is used to direct automatic level shifter insertion to insert level shifter on specified clocks

TYPE

string

DEFAULT

""

GROUP

none

DESCRIPTION

This variable can be set to "all" or a list of clock names (delimited by space or comma). When set to "all", automatic level shifter insertion will insert level shifters on all clock nets that need level shifters.

When this variable is set to a list of clock names, automatic level shifter insertion will insert level shifters on the net of these clocks if needed.

By default automatic level shifter insertion will not insert level shifters on the nets driven by the clocks.

auto_link_disable

Specifies whether the code to perform an auto_link during any Design Compiler command should be disabled.

TYPE

Boolean

DEFAULT

false

GROUP

system_variables

DESCRIPTION

When true, specifies that the code to perform an auto_link during any Design Compiler command should be disabled, resulting in a speedup in command processing. This speedup is important in backannotation commands like **set_load**, **set_resistance**, and **set_annotated_delay** where potentially thousands of such commands are executed in sequence. Disabling the auto_link code can significantly improve the speed with which such commands get executed.

Once the sequence of time-critical commands has been completed, this variable should be reset to the value false, to revert the Design Compiler back to its normal mode of operation.

To determine the current value of this variable use **printvar auto_link_disable**. For a list of all **system** variables and their current values, use the **print_variable_group system** command.

SEE ALSO

set_load(2)
set_resistance(2)
set_annotated_delay(2)

auto_link_options

Specifies the **link** command options to be used when **link** is invoked automatically by various Design Compiler and DFT Compiler commands (for example, **create_schematic** and **compile**).

TYPE

string

DEFAULT

-all

GROUP

system_variables

DESCRIPTION

Specifies the **link** command options to be used when **link** is invoked automatically by various Design Compiler and DFT Compiler commands (for example, **create_schematic** andcompile). The default is **-all**. To find the available options, refer to the **link** command manual page.

To determine the current value of this variable, type **printvar auto_link_options**. For a list of all **system** variables and their current values, type **print_variable_group system**.

SEE ALSO

link(2)

auto_ungroup_preserve_constraints

Preserves (when *true*) the timing constraints on the hierarchy when the hierarchy is ungrouped during the process of optimization.

TYPE

Boolean

DEFAULT

true

GROUP

timing

DESCRIPTION

Enables the ungrouping (when *true*) of hierarchies with timing constraints and preserves the timing constraints when the hierarchies are ungrouped during the process of optimization By default, the **auto_ungroup_preserve_constraints** variable is *true*,

The constraints are preserved when executing the following commands:

compile -ungroup_all

set_ungroup followed by **compile**

compile -auto_ungroup area | delay

The constraints on DesignWare library hierarchical cells are also lost when the cells are ungrouped during optimization.

Set the **auto_ungroup_preserve_constraints** variable to *false* before compiling to avoid ungrouping of hierarchies with timing constraints.

Use the **printvar auto_ungroup_preserve_constraints** command to determine the current value of this variable.

SEE ALSO

compile(2)
set_ungroup(2)

auto_wire_load_selection

Controls automatic selection of wire load model.

TYPE

string

DEFAULT

area_locked

GROUP

compile_variables

DESCRIPTION

When area_locked, the automatic wire load selection uses the initial area to do the first selection of the wire load model and then adjusts the wire load model down if the area drops. When area_reselect, the automatic wire load selection during reporting and at various points in compile updates the wire load model to the current area of the design. When false, the automatic wire load selection is off. For backwards compatibility, we also support the value true. True is the same as area_locked.

The automatic selection of the wire load model is used to estimate net capacitances and resistances from the net fanout. The wire load models are described in the technology library. With the automatic selection of the wire load model, if the wire load mode is **segmented** or **enclosed**, the wire load model will be chosen based on the area of the design containing the net either partially (for **segmented**) or fully (for **enclosed**). If the wire load mode is **top**, the wire load model will be chosen based on the area of the top level design for all nets in the design hierarchy.

When a design's wire load model is selected manually by the user (with the command **set_wire_load**), no wire load is selected automatically for that design.

To determine the current value of this variable use **printvar auto_wire_load_selection**. For a list of all **compile** variables and their current values, use the **print_variable_group compile** command.

SEE ALSO

set_wire_load (2), **report_design** (2); **compile_variables** (3)

bind_unused_hierarchical_pins

Specifies if unused input and output hierarchical pins should be connected to constant tie-off cells during compile.

TYPE

Boolean

DEFAULT

true

GROUP

compile_variables

DESCRIPTION

Design Compiler connects each undriven leaf cell input to a constant tie-off cell during compile. The tool does the same for input pins and output pins of a design hierarchy as well. If an input pin of a design hierarchy is undriven, it will be connected to a constant tie-off cell in its parent hierarchy during compile. If an output pin of a design hierarchy is undriven, it will be connected to a constant tie-off cell in the same hierarchy during compile.

To prevent unconnected pins from being tied off, set **bind_unused_hierarchical_pins** to false before linking in the design.

When commands such as **compile -boundary_optimization** are used, the optimized design may be left with hierarchical inputs and outputs that are neither driven nor loaded. These hierarchical inputs and outputs will not be connected to constant tie-offs if **bind_unused_hierarchical_pins** variable is set to false.

To determine the current value of this variable, type **printvar bind_unused_hierarchical_pins**. For a list of all **compile** variables and their current values, type **print_variable_group compile**.

SEE ALSO

compile(2)

bound_attributes

Contains attributes related to bound.

DESCRIPTION

Contains attributes related to bound.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified object. Specified with **list_attribute -class bound -application**, the definition of attributes can be listed.

Bound Attributes

aspect_ratio
: Specifies the **width**:**height** ratio of a bound.
The data type of **aspect_ratio** is double.
This attribute is read-only.

bbox
: Specifies the bounding-box of a bound. The **bbox** is represented by a **rectangle**.
The format of a *rectangle* specification is {{llx lly} {urx ury}}, which specifies the lower-left and upper-right corners of the rectangle.
The data type of **bbox** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_ll
: Specifies the lower-left corner of the bounding-box of a bound.
The **bbox_ll** is represented by a **point**. The format of a *point* specification is {x y}.
You can get the **bbox_ll** of a bound, by accessing the first element of its **bbox**.
The data type of **bbox_ll** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_llx
: Specifies x coordinate of the lower-left corner of the bounding-box of a bound.
The data type of **bbox_llx** is double.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_lly
: Specifies y coordinate of the lower-left corner of the bounding-box of a bound.
The data type of **bbox_lly** is double.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_ur

Specifies the upper-right corner of the bounding-box of a bound.
The **bbox_ur** is represented by a **point**. The format of a *point* specification is {x y}.
You can get the **bbox_ur** of a bound, by accessing the second element of its **bbox**.
The data type of **bbox_ur** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_urx

Specifies x coordinate of the upper-right corner of the bounding-box of a bound.
The data type of **bbox_urx** is double.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_ury

Specifies y coordinate of the upper-right corner of the bounding-box of a bound.
The data type of **bbox_ury** is double.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

color

Specifies color to draw a move bound and its associated instances.
The data type of **color** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

dimension

Specifies dimension of a group bound.
Its format is {**width height**}.
The data type of **dimension** is string.
This attribute is read-only.

effort

Specifies effort to bring cells closer inside an auto group bound.
Its valid values can be:

- **low**
- **medium**
- **high**
- **ultra**

The data type of **effort** is string.

This attribute is read-only.

name

Specifies name of a bound object.
The data type of **name** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

number_of_hard_macro

Specifies number of hard macro cells inside a bound.
The data type of **number_of_hard_macro** is integer.
This attribute is read-only.

number_of_standard_cell

Specifies number of standard cells inside a bound.
The data type of **number_of_standard_cell** is integer.
This attribute is read-only.

object_class

Specifies object class name of a bound, which is **bound**.
The data type of **object_class** is string.
This attribute is read-only.

object_type

Specifies object type of a bound, which can be **move_bound**, **auto_group_bound**, or **group_bound**.
The data type of **object_type** is string.
This attribute is read-only.

points

Specifies point list of a move bound’s boundary.
The data type of **points** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

type

Specifies type of a move bound or a group bound.
The data type of **type** is string.
Its valid values are:

- **soft**
- **hard**
- **exclusive**

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

utilization

    Specifies the ratio of total area size of associated instances to the area of a move bound.
    The data type of **utilization** is double.
    This attribute is read-only.

SEE ALSO

get_attribute(2),
list_attribute(2),
report_attribute(2),
set_attribute(2).

bsd_max_in_switching_limit

Specifies the maximum number of design inputs that may switch simultaneously while generating input DC parametric tests using the **create_bsd_patterns** command.

TYPE

integer

DEFAULT

60000

GROUP

bsd_variables

DESCRIPTION

Switching all inputs and outputs simultaneously can cause large currents. It is advisable to switch as few I/Os as possible while performing DC parametric tests. The **bsd_max_in_switching_limit** variable lets you specify the maximum number of inputs that may switch simultaneously while generating VIL/VIH tests using the **create_bsd_patterns** command.

To determine the current value of this variable, type **printvar bsd_max_in_switching_limit**. For a list of **bsd** variables and their current values, type **print_variable_group bsd**.

SEE ALSO

create_bsd_patterns(2)

bsd_max_out_switching_limit

Specifies the maximum number of design outputs that may switch simultaneously while generating output DC parametric tests using the **create_bsd_patterns** command.

TYPE

integer

DEFAULT

60000

GROUP

bsd_variables

DESCRIPTION

Switching all inputs and outputs simultaneously can cause large currents. It is advisable to switch as few I/Os as possible while performing DC parametric tests. The **bsd_max_out_switching_limit** variable lets you specify the maximum number of outputs that may switch simultaneously while generating VOL/VOH tests using the **create_bsd_patterns** command.

To determine the current value of this variable, type **printvar bsd_max_out_switching_limit**. For a list of **bsd** variables and their current values, type **print_variable_group bsd**.

SEE ALSO

create_bsd_patterns(2)

```
Specifies the effort used by physical compiler
```

bsd_physical_effort

TYPE

string

DEFAULT

medium

GROUP

bsd_variables

DESCRIPTION

Specifies the effort used for BSD compiler and Physical Compiler integration features.

Allowable values for this variable are *low*, *medium*, *high*, and *ultra*. The default value is *medium*.

To determine the current value of this variable, type **printvar bsd_physical_effort**. For a list of **bsd** variables and their current values, type **print_variable_group bsd**.

SEE ALSO

bsd_variables

SYNTAX

integer **test_bsd_allow_tolerable_violations** = FALSE
integer **test_bsd_control_cell_drive_limit** = 3
integer **test_bsd_manufacturer_id** = 0
integer **test_bsd_optimize_control_cell** = FALSE
integer **test_bsd_part_number** = 0
integer **test_bsd_version_number** = 0
string **test_bsdl_default_suffix_name** = "bsdl"
integer **test_bsdl_max_line_length** = 80
integer **test_cc_ir_masked_bits** = 0
integer **test_cc_ir_value_of_masked_bits** = 0
string **test_user_defined_instruction_naming_style** = "USER%d"
string **test_user_defined_test_data_register_naming_style** = "UTDR%d"
string **bsd_max_in_switching_limit** = 60000
string **bsd_max_out_switching_limit** = 60000

DESCRIPTION

These variables directly affect the **insert_bsd**, **optimize_bsd**, **check_bsd**, **create_bsd_patterns** and **write_bsdl** commands. Defaults are shown above, under Syntax.

For a list of **bsd** variables, type **print_variable_group bsd.** To view this manual page on-line, type **help bsd_variables.** To view an individual variable description, type **help *variable-name***, where *variable-name* is the name of the variable.

test_bsd_allow_tolerable_violations
When *true*, allows **optimize_bsd** to exchange **control and observe** cells for **observe only** cells, or to remove the cells altogether, thereby allowing tolerable violations for timing-driven optimization. The default is *false*.

test_bsd_control_cell_drive_limit
An integer that specifies the number of output bits to be controlled by a single BSR controlling cell. This controls the assignment of BSR control cells during **optimize_bsd.**

test_bsd_manufacturer_id
An integer that specifies the **manufacturer id** to be part of the Device Identification Register's capture value during **insert_bsd.**

test_bsd_optimize_control_cell
When *true*, enables control cell optimization during **optimize_bsd.** The default is *false*. The number of cells controlled by a single BSR cell is controlled by the **test_bsd_control_cell_drive_limit** variable.

test_bsd_part_number

An integer that specifies the **Part Number** to be part of the Device Identification

Register's capture value during **insert_bsd**.

test_bsd_version_number
An integer that specifies the **Version Number** to be part of the Device Identification Register's capture value during **insert_bsd**.

test_bsdl_default_suffix_name
Specifies the default suffix for name of the the BSDL file generated by the **write_bsdl** command.

test_bsdl_max_line_length
An integer that specifies the maximum number of characters per line of the output BSDL file produced by the **write_bsdl** command.

test_cc_ir_masked_bits
An integer whose binary value specifies instruction register (IR) bits that are to be masked during the search for all possible implemented instructions. Bits containing a "1" value are masked.

test_cc_ir_value_of_masked_bits
An integer that specifies a value (0 or 1) to be forced into the instruction register (IR) bits masked by the **test_cc_ir_masked_bits** variable.

test_user_defined_instruction_naming_style
Specifies the naming style to be used by **check_bsd** and **write_bsdl** for the user-defined (non-standard) instructions inferred by these commands.

test_user_test_data_register_naming_style
Specifies the naming style to be used by **check_bsd** and **write_bsdl** for the user-defined (non-standard) test data registers inferred by these commands.

bsd_max_in_switching_limit
Specifies the maximum number of inputs that may switch simultaneously while generating VIL/VIH tests using the **create_bsd_patterns** command.

bsd_max_out_switching_limit
Specifies the maximum number of outputs that may switch simultaneously while generating VOL/VOH tests using the **create_bsd_patterns** command.

SEE ALSO

insert_bsd (2), **optimize_bsd** (2), **check_bsd** (2), **write_bsdl** (2), **create_bsd_patterns** (2), **test_bsd_version_number** (3), **test_bsd_manufacturer_id** (3), **test_bsd_part_number** (3), **test_bsd_optimize_control_cell** (3), **test_bsd_control_cell_drive_limit** (3), **test_bsd_allow_tolerable_violations** (3). **test_cc_ir_masked_bits** (3), **test_cc_ir_value_of_masked_bits** (3), **test_bsdl_default_suffix_name** (3), **test_bsdl_max_line_length** (3), **test_user_defined_instruction_naming_style** (3), **test_user_test_data_register_naming_style** (3).

budget_generate_critical_range

Enables automatic generation of set_critical_range commands by dc_allocate_budgets for multiply-instantiated subdesigns.

TYPE

Boolean

DEFAULT

false

GROUP

acs_variables

DESCRIPTION

When set to true, this variable will cause the **dc_allocate_budgets** command to automatically generate a **set_critical_range** command for any budgeted cells that have multiply-instantiated subdesigns. The value of critical_range is set to 10% of the shortest clock period in the design. If the top-level design already has a critical_range attribute, then that original value will be used instead.

The purpose of this is to improve QoR, since any multiple instances must be dont_touched after compile in a bottom-up compile flow. Using **set_critical_range** will cause any near-critical paths in these blocks to be optimized before the block is dont_touched.

To see the current value of this variable, type

```
dc_shell-xg-t> printvar budget_generate_critical_range
```

SEE ALSO

dc_allocate_budgets(2)

budget_map_clock_gating_cells

Maps integrated clock gating cells into target library during RTL budgeting.

TYPE

Boolean

DEFAULT

false

GROUP

acs_variables

DESCRIPTION

When set to true, this variable will cause the **dc_allocate_budgets -mode rtl** command to map any integrated clock gating cells into target library cells before calculating the budget. Normally, any unmapped cells would remain unmapped during budgeting when **-mode rtl** is used.

The purpose of this is to prevent incorrect **-level_sensitive** delay constraints from appearing in the budget constraint files. These may otherwise appear because unmapped integrated clock gating cells can include transparent latch elements.

To see the current value of this variable, type

```
dc_shell-xg-t> printvar budget_map_clock_gating_cells
```

SEE ALSO

dc_allocate_budgets(2)

bus_inference_descending_sort

Specifies that the members of that port bus are to be sorted in descending order rather than in ascending order.

TYPE

Boolean

DEFAULT

true

GROUP

edif_variables
io_variables

DESCRIPTION

Affects the **read** command except for the db, Verilog and VHDL formats. This variable is primarily used when reading in designs in the LSI/NDL format. That particular format does not support representation of busses, but, if port names follow a specific pattern (as described in the variable **bus_inference_style**), the individual bits can be "inferred" into a port bus. When true (the default value), This variable specifies that the members of that port bus are to be sorted in descending order rather than in ascending order.

For example, with the variable **bus_inference_style** set to "%s[%d]", the ports "A[1]", "A[2]", "A[3]", and "A[4]" will be "inferred" into a port bus named "A". If this variable is true, the port bus "A" will have an index from 4 to 1; if this variable is false, the port bus "A" will have an index from 1 to 4.

With the variable **bus_inference_style** set to "#%d%%%s", the ports "#8%cb", "#9%cb", "#10%cb", and "#11%cb" will be "inferred" into a port bus named "cb". If this variable is true, the port bus "cb" will have an index from 11 to 8; if this variable is false, the port bus "cb" will have an index from 8 to 11.

To determine the current value of this variable, type **printvar bus_inference_descending_sort**. For a list of all **edif** or **io** variables and their current values, type **print_variable_group edif** or **print_variable_group io**.

SEE ALSO

create_bus(2)
remove_bus(2)
report_bus(2)
bus_inference_style(3)
bus_minus_style(3)
bus_naming_style(3)
bus_range_separator_style(3)

```
edif_variables(3)
io_variables(3)
```

bus_inference_style

Specifies the pattern used to infer individual bits into a port bus.

TYPE

string

DEFAULT

""

GROUP

edif_variables
io_variables

DESCRIPTION

Specifies the pattern used to infer individual bits into a port bus. This variable affects the **read** command except, for the db and VHDL formats. This variable also affects the VHDL **write** commands. The variable is used primarily when reading in designs in the LSI/NDL format. The LSI/NDL format does not support representation of buses. But if port names follow a specific pattern (as described by this variable), the individual bits can be "inferred" into a port bus. If you specify an invalid value, no port buses are inferred.

When running in tcl mode the bus_inference_style needs to be within the curly brackets and if running in eqn mode, the bus_inference_style needs to be within double quotes.

For example: eqn mode: bus_inference_style = "%s[%d]"

tcl mode: set bus_inference_style {%s[%d]}

This variable must contain one %s (percent s) and %d (percent d) character sequence. Additional characters can be used with these symbols. To use a percent sign in a name, two are needed in the variable string (%%).

In naming a port bus, the port name is substituted for %s, and the port number replaces %d. A single percent sign is substituted for %%.

For example, with this variable set to "%s[%d]", the ports "A[1]", "A[2]", "A[3]", and "A[4]" will be "inferred" into a port bus named "A" either with an index from 1 to 4 or with an index from 4 to 1 (see **bus_inference_descending_sort**).

With this variable set to "#%d%%%s", the ports "#8%cb", "#9%cb", "#10%cb", and "#11%cb" will be "inferred" into a port bus named "cb" either with an index from 8 to 11 or with an index from 11 to 8 (see **bus_inference_descending_sort**).

To determine the current value of this variable, type **printvar bus_inference_style**.

For a list of all **edif** or **io** variables and their current values, type **print_variable_group edif** or **print_variable_group io**.

SEE ALSO

```
create_bus(2)
remove_bus(2)
report_bus(2)
bus_inference_descending_sort(3)
bus_minus_style(3)
bus_naming_style(3)
bus_range_separator_style(3)
edif_variables(3)
io_variables(3)
```

bus_minus_style

Controls the naming of individual members of bit-blasted port, instance, or net buses with negative indices.

TYPE

string

DEFAULT

-%d

GROUP

hdl_variables

DESCRIPTION

Controls the naming of individual members of bit-blasted port, instance, or net buses with negative indices. This variable affects the **read** command with the **vhdl** format option.

To determine the current value of this variable, type **printvar bus_minus_style**. For a list of all **hdl** variables and their current values, type **print_variable_group hdl**.

SEE ALSO

create_bus(2)
remove_bus(2)
report_bus(2)
bus_inference_descending_sort(3)
bus_inference_style(3)
bus_naming_style(3)
bus_range_separator_style(3)
edif_variables(3)

bus_multiple_separator_style

Determines the name of a multibit cell that implements bits that do not form a range.

TYPE

string

DEFAULT

,

GROUP

multibit_variables

DESCRIPTION

This variable affects the naming of multibit cells during multibit mapping. This variable is used to name a multibit cell that implements bits that do not form a range. The default is used if an invalid value is specified.

The **bus_range_separator_style** is used to separate the start and end bit positions of a range, while **bus_multiple_separator_style** is used to separate two ranges. The two variables are used in conjunction with the **bus_naming_style** variable to generate names for multibit cells.

Assume that **bus_range_separator_style** is set to ":", **bus_multiple_separator_style** is set to ",", and **bus_naming_style** is set to "%s[%d] in the following examples.

For example, if cells with the names q[0], q[1], q[2], q[5], q[6], and q[7] are packed into a 6-bit wide cell, the name given to the new cell is q[0:2,5:7]. If cells q[0], q[2], q[4], and q[6] are packed into a 4-bit wide cell, the name given to the new cell is q[0,2,4,6].

To determine the current value of this variable, type **printvar bus_range_separator_style**. For a list of all **multibit** variables and their current values, type **print_variable_group multibit**.

SEE ALSO

bus_range_separator_style(3)
multibit_variables(3)

bus_naming_style

Specifies the style to use in naming an individual port member, net member, or cell instance member of an EDIF array or of a Verilog or VHDL vector.

TYPE

string

DEFAULT

%s[%d]

GROUP

edif_variables
hdl_variables
schematic_variables

DESCRIPTION

This variable affects the **read** command with the EDIF, Verilog, or VHDL format option, the **write** command with the EDIF format option, and the **create_schematic** command with the busing option.

When reading buses, this variable specifies the style to use in naming an individual port member, net member, or cell instance member of an EDIF array or of a Verilog or VHDL vector.

When running in tcl mode the bus_naming_style needs to be within the curly brackets and if running in eqn mode, the bus_naming_style needs to be within double quotes.

For example: eqn mode: bus_naming_style = "%s[%d]"

tcl mode: set bus_naming_style {%s[%d]}

When writing buses, this variable used with the variable **bus_range_separator_style** specifies the style to use in naming a port array or net array in the EDIF file. If you specify an invalid value, the array is given the name of the bus. When writing schematic nets, this variable used with the variable **bus_range_separator_style** specifies the style to use in naming a net connected to the "wire" end of a ripper in the EDIF file. If you specify an invalid value, the net is given the name of the original net.

When creating schematics, this variable used with the variable **bus_range_separator_style** specifies the style to use in naming a ripper, bused port, bused net, or net connected to the "wire" end of a ripper. If you specify an invalid value, the default is used.

This variable must contain only one %s (percent s) and one %d (percent d) character sequence. To use the % (percent sign) in the name, use two of them in the variable string (%%). Therefore, the only characters that can follow a percent sign are %, s, or d.

When reading buses, in naming members, the name of the array is substituted for %s, and the number of the member is substituted for %d. A single percent sign is substituted for %%.

For example, if this variable is set to "%s[%d]", then the first member of the 4-bit array "A", going from 0 to 3, is named:

A[0]

If this variable is set to "%s_%%%d.X", then the first member of the 8-bit array "xy", going from 9 to 16, is named:

xy_%9.X

See **bus_dimension_separator_style** for a description of how it is used in conjunction with this variable for specifying the names of the members of multi-dimensional arrays in the EDIF format or multi-dimensional vectors in the VHDL format.

See **bus_minus_style** for a description of how to specify the names of vectors with negative indices in the VHDL format.

When creating schematics or when writing buses, in naming a bused port or bused net or a port array or net array, the original bus or array name is substituted for %s. The start and end bits of the bus or array separated by the value of the variable **bus_range_separator_style** are substituted for %d. A single percent sign is substituted for each %%.

For example, if this variable is set to "%s[%d:%d]", then the 4-bit bus or array "A", going from 0 to 3, is named:

A[0:3]

If this variable is set to "%%%s[%d][%d]", then the 8-bit bus or array "B", going from -4 to 3, is named:

See **edifout_multidimension_arrays** for a description of how it is used in conjunction with this variable for specifying the names of multi-dimensional arrays.

See **edifout_numerical_array_members** for a description of how it is used in conjunction with this variable for specifying the names of descending arrays.

When creating schematics or when writing schematic nets, in naming a ripper or net connected to the "wire" end of a ripper, the original net name is substituted for %s. If the net is a scalar (a single bit) net, the bit of the ripper is substituted for %d; if the net is a bused net, the start and end bits of the ripper separated by the value of the variable **bus_range_separator_style** are substituted for %d. A single percent sign is substituted for each %%.

For example, if this variable is set to "%s[%d]", then the ripper or the net connected to the "wire" end of the ripper that is ripping off the first bit of the 4-bit net array "A" going from 0 to 3, is named:

```
A[0]
```

If this variable is set to "%s_%%%d.X", then the ripper or the net connected to the "wire" end of the ripper that is ripping off the first bit of the 8-bit net array "xy" going from 9 to 16, is named:

```
xy_%9.X
```

If this variable is set to "%s[%d]" and the variable **bus_range_separator_style** is set to ":", then the net connected to the "wire" end of the ripper that is ripping off the third through fifth bits of the 8-bit net array "xy" going from 9 to 16, is named:

```
xy[11:13]
```

If this variable is set to "%s_%%%d.X" and the variable **bus_range_separator_style** is set to "..", then the ripper or the net connected to the "wire" end of the ripper that is ripping off the third through fourth bits of the 4-bit net array "A" going from 0 to 3, is named:

```
A_%2..3.X
```

To determine the current value of this variable, type **printvar bus_naming_style**. For a list of all **edif**, **hdl**, or **schematic** variables and their current values, type **print_variable_group edif**, **print_variable_group hdl**, or **print_variable_group schematic**.

SEE ALSO

create_bus(2)
remove_bus(2)
report_bus(2)
bus_inference_descending_sort(3)
bus_inference_style(3)
bus_minus_style(3)
bus_range_separator_style(3)
edif_variables(3)

bus_range_separator_style

Specifies the style to use in naming a net connected to the "wire" end of a ripper in the EDIF file.

TYPE

string

DEFAULT

:

GROUP

edif_variables
schematic_variables

DESCRIPTION

Specifies the style to use in naming a net connected to the "wire" end of a ripper in the EDIF file. This variable affects the **write** command with the EDIF format option and the **create_schematic** command with the busing option.

When writing buses, this variable used with the variable **bus_naming_style**, specifies the style to use in naming a port array or net array in the EDIF file. When writing schematic nets, this variable used with the variable **bus_naming_style**, specifies the style to use in naming a net connected to the "wire" end of a ripper in the EDIF file.

When creating schematics, this variable used with the variable **bus_naming_style**, specifies the style to use in naming a ripper, bused port, bused net, or net connected to the "wire" end of a ripper. If you specify an invalid value, the default is used.

See **bus_naming_style** for a description of how this variable is used in conjunction with that variable.

To determine the current value of this variable, type **printvar bus_range_separator_style**. For a list of all **edif** or **schematic** variables and their current values, type **print_variable_group edif** or **print_variable_group schematic**.

SEE ALSO

create_bus(2)
remove_bus(2)
report_bus(2)
bus_inference_descending_sort(3)
bus_inference_style(3)
bus_minus_style(3)
bus_naming_style(3)

cache_dir_chmod_octal

Specifies the value of the mode bits for created cache directories.

TYPE

string

DEFAULT

777

GROUP

synlib_variables

DESCRIPTION

Cache directories are created with their mode bits set to the value of the **cache_dir_chmod_octal** variable. The value of this variable is a string that is translated to an octal number. There are separate variables for directories and files, to allow the sticky bit to be set.

Many UNIX systems allow a sticky bit to be set on directories (setting the sticky bit on files has a very different meaning and is not allowed for cache files). If the sticky bit on a directory is set and if the user has write permission on the directory, the user can write his or her own files in the directory but cannot delete the files of other people in that directory (see UNIX manpage on sticky(8)).

Caches are often shared among users, and the sticky bit allows users to write into the same directory without worrying that other users will overwrite their files. If your UNIX system does not have sticky bit capabilities, your system administrator should remove that bit from the cache-dir-chmod-octal default in the system **.synopsys_dc.setup** file.

SEE ALSO

cache_ls(1)
cache_write(3)
cache_file_chmod_octal(3)
cache_read(3)

cache_file_chmod_octal

Specifies the value of the mode bits for created cache files.

TYPE

string

DEFAULT

666

GROUP

synlib_variables

DESCRIPTION

Cache files are created with their mode bits set to the value of cache_file_chmod_octal. The value of this variable is a string which is translated to an octal number. Cache directories use a different variable to set there mode bits (cache_dir_chmod_octal). There are separate variables for directories and files to allow the sticky bit to be set.

SEE ALSO

cache_ls(1)
cache_dir_chmod_octal(3)
cache_read(3)
cache_write(3)

cache_read

Specifies a list of directories that contain cache files that will be read from whenever a cache entry is needed.

TYPE

list

DEFAULT

~

GROUP

synlib_variables

DESCRIPTION

Specifies a list of directories that contain cache files that will be read from whenever a cache entry is needed. If the variable is set to an empty list "{}", there are no directories containing caches and caching is effectively turned off. A Synopsys cache is stored as a UNIX directory tree rooted in one of the directories listed in the **cache_read** variable.

To determine the current value of this variable, type **printvar cache_read**. For a list of **synlib** variables and their current values, type **print_variable_group synlib**.

SEE ALSO

cache_read_info(3)
cache_write(3)
synlib_variables(3)

cache_read_info

Specifies whether an informational message will be printed each time a cache element is read.

TYPE

Boolean

DEFAULT

false

GROUP

synlib_variables

DESCRIPTION

Specifies whether an informational message will be printed each time a cache element is read.

SEE ALSO

cache_ls(1)
cache_read(3)
cache_write_info(3)

cache_write

Specifies the directory where optimized and unoptimized synlib parts will be written, if they are not already in the cache.

TYPE

string

DEFAULT

~

GROUP

synlib_variables

DESCRIPTION

Specifies the directory where optimized and unoptimized synlib parts will be written, if they are not already in the cache (see cache_read(3)). If the variable is set to an empty string (""), then synlib parts will not be written to the cache. A Synopsys cache is stored as a UNIX directory tree rooted in the directory listed in the **cache_write** variable.

To determine the current value of this variable, type **printvar cache_write**. For a list of **synlib** variables and their current values, type **print_variable_group synlib**.

SEE ALSO

cache_read(3)
cache_read_info(3)
cache_write_info(3)
cache_file_chmod_octal(3)
cache_dir_chmod_octal(3)
synlib_variables(3)

cache_write_info

Specifies whether an informational message will be printed each time a cache element is written.

TYPE

Boolean

DEFAULT

false

GROUP

synlib_variables

DESCRIPTION

Specifies whether an informational message will be printed each time a cache element is written. If true (the default), an information message will be printed each time a cache element is written.

SEE ALSO

cache_write(3)
cache_read_info(3)

case_analysis_log_file

Specifies the name of a log file generated during propagation of constant values, from case analysis or from nets tied to logic zero or logic one. Each scenario has its proprietary log file if multiple scenarios exist.

TYPE

string

DEFAULT

""

DESCRIPTION

Specifies the name of a log file generated during propagation of constant values, from case analysis or from nets tied to logic zero or logic one. The log file contains the list of all ports and pins that propagate constants. The constant propagation algorithm is an iterative process that propagates constants through nets and cells starting from a list of constant pins. The algorithm finishes when no more constants can be propagated. The format of the log file follows the constant propagation algorithm.

In MCMM, you need to specify the log file name in the definition of each scenario, or no log will be generated for that scenario. If you switch the active scenario, the log file used will also be switched automatically. Log file for different scenarios can have the same name.

By default, this variable is set to an empty string, and no log file is generated during constant propagation.

To determine the current value of this variable, use **printvar case_analysis_log_file**. Please note that if you switch the scenario in MCMM, the value of this variable will remain to the value last set.

SEE ALSO

remove_case_analysis(2)
report_case_analysis(2)
report_disable_timing(2)
set_case_analysis(2)
disable_case_analysis(3)

case_analysis_propagate_through_icg

Determines whether case analysis is propagated through integrated clock gating cells.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When *false* (the default), constants propagating throughout the design will stop propagating when an integrated clock gating cell is encountered. Regardless of whether the integrated clock gating cell is enabled or disabled, no logic values will propagate in the fanout of the cell.

When *true*, constants propagated throughout the design will propagate through an integrated clock gating cell provided the cell is enabled. An integrated clock gating cell is enabled when its enable pin (or test enable pin) is set to a hi logic value. If the cell is disabled, then the disable logic value for the cell is propagated in its fanout. e.g. for a latch_posedge ICG, when it is disabled, it will propagate a logic 0 in its fanout.

To activate logic propagation through all integrated clock gating cells, the user must set the following.

```
set case_analysis_propagate_through_icg true
```

To determine the current value of this variable, type **printvar case_analysis_propagate_through_icg** or **echo $case_analysis_propagate_through_icg**.

SEE ALSO

set_case_analysis(2)
remove_case_analysis(2)

case_analysis_with_logic_constants

When true, enables constant propagation, even if a design contains only logic constants.

TYPE

Boolean

DEFAULT

false

GROUP

timing

DESCRIPTION

When true, enables constant propagation, even if a design contains only logic constants. When false (the default), constant propagation is not performed unless a **set_case_analysis** command is specified. The variable **disable_case_analysis** overrides the variable **case_analysis_with_logic_constants**. If the **disable_case_analysis** variable is set, no constants are propagated.

To determine the current value of this variable, use **printvar case_analysis_with_logic_constants**.

SEE ALSO

remove_case_analysis(2)
report_case_analysis(2)
set_case_analysis(2)
disable_case_analysis(3)

cell_attributes

Contains attributes that can be placed on a cell.

DESCRIPTION

Contains attributes that can be placed on a cell.

There are a number of commands used to set attributes, however, most attributes can be set with the **set_attribute** command. If the attribute definition specifies a **set** command, use it to set the attribute. Otherwise, use **set_attribute**. If an attribute is read-only, you cannot set it.

To determine the value of an attribute, use the **get_attribute** command. To remove attributes, use the **remove_attribute** command.

For a more detailed explanation of an attribute, refer to the manual pages of the appropriate **set** command. For information on all attributes, refer to the **attributes** manual page.

Cell Attributes

area

Specifies the area of a cell. This attribute does not exist on a hierarchical cell.
The tool calculates the attribute by the cell's boundary points.
This attribute is read-only and cannot be modified.

aspect_ratio

Specifies the **height:width** ratio of a cell. This attribute does not exist on a hierarchical cell.
This attribute is read-only and cannot be modified.

async_set_reset_q

Establishes the value (0 or 1) that should be assigned to the q output of an inferred register if set and reset are both active at the same time. To be used with **async_set_reset_qn**. Use these attributes only if you have used the **one_hot** or **one_cold** attributes/directives in your HDL description *and* your technology library is written using pre-V3.0a syntax; *or* if your technology library does not use a consistent convention for q and qn when set and reset are both active. If a V3.0a or later syntax technology library is used, then by default if set and reset are both active at the same time Design Compiler will use the convention of the selected technology library (**target_library**).
Set with **set_attribute**.
Note: If you are unsure whether or not your technology library uses V3.0a syntax, ask your ASIC vendor.

async_set_reset_qn

Establishes the value (0 or 1) that should be assigned to the qn output of an inferred register if set and reset are both active at the same time. To be used with **async_set_reset_q**. Use these attributes only if you have used the **one_hot** or **one_cold** attributes/directives in your HDL description *and* your technology library is written using pre-V3.0a syntax; *or* if your

technology library does not use a consistent convention for q and qn when set and reset are both active. If a V3.0a or later syntax technology library is used, then by default if set and reset are both active at the same time Design Compiler will use the convention of the selected technology library (**target_library**). Set with **set_attribute**.
Note: If you are unsure whether or not your technology library uses V3.0a syntax, ask your ASIC vendor.

disable_timing *
Disables the timing arcs of a cell. This has the same effect on timing as not having the arc in the library. Set with fbset_disable_timing.

dont_touch
Identifies cells to be excluded from optimization. Values are *true* (the default) or *false*. Cells with the **dont_touch** attribute set to *true* are not modified or replaced during **compile**. Setting **dont_touch** on a hierarchical cell sets the attribute on all cells below it. Set with **set_dont_touch**.

fall_delay
Specifies an offset from the falling edge of the ideal clock waveform. Affects all clock pins on this cell. Set with **set_clock_skew -fall_delay**.

flip_flop_type
Stores the name of the specified flip-flop to be converted from the **target_library**. The **compile** command automatically converts all tagged flip-flops to the specified (or one similar) type. Set with **set_register_type -flip_flop** *flip_flop_name [cell_list]*.

flip_flop_type_exact
Stores the name of the specified flip-flop to be converted from the **target_library**. The **compile** command automatically converts all tagged flip-flops to the exact flip-flop type. Set with **set_register_type -exact -flip_flop** *flip_flop_name [cell_list]*.

height
Specifies the height of a cell. This attribute does not exist on a hierarchical cell.
The tool uses the cell’s cell boundary to calculate its height.
This attribute is read-only and cannot be modified.

is_black_box
Set to *true* if the cell’s reference is not linked to a design.
This attribute is read-only and cannot be modified.

is_combinational
Set to *true* if all cells of a design and all designs in its hierarchy are combinational. A cell is combinational if it is non-sequential or non-tristate and all of its outputs compute a combinational logic function. The **report_lib** command will report such a cell as not a black-box.
This attribute is read-only and cannot be modified.

is_dw_subblock
Set to *true* if the object (a cell, a reference, or a design) is a DW subblock that was automatically elaborated.
This attribute is read-only and cannot be modified.

Note: DW subblocks that are manually elaborated will not have this attribute.

is_hierarchical
: Set to *true* if the design contains leaf cells or other levels of hierarchy. This attribute is read-only and cannot be modified.

is_mapped
: Set to *true* if the cell is not generic logic.
This attribute is read-only and cannot be modified.

is_sequential
: Set to *true* if the cell is sequential. A cell is sequential if it is not combinational.
This attribute is read-only and cannot be modified.

is_synlib_module
: Set to *true* if the object (a cell, a reference, or a design) refers to an unmapped module reference or if the object is (or refers to) a design that was automatically elaborated from a synlib module or a synlib operator.
This attribute is read-only and cannot be modified.
Note: synlib modules that are manually elaborated will not have this attribute.

is_synlib_operator
: Set to *true* if the object (a cell or a reference) is a synthetic library operator reference.
This attribute is read-only and cannot be modified.

is_test_circuitry
: Set by **insert_dft** on the scan cells and nets added to a design during the addition of test circuitry.
This attribute is read-only and cannot be modified.

is_unmapped
: *true* if the cell is generic logic.
This attribute is read-only and cannot be modified.

latch_type_exact
: Stores the name of the specified latch to be converted from the **target_library**. The **compile** command automatically converts all tagged latches to the exact latch type. Set with **set_sequential_type -latch** *latch_name [cell_list].*

macro_area_percentage
: Specifies the percentage of total macro area of a soft macro‚Äôs physical_area
This attribute is read-only and cannot be modified.

map_only
: When set to *true*, **compile** will attempt to map the object exactly in the target library, and will exclude the object from logic-level optimization (flattening and structuring). The default is *false*. Set with **set_map_only**.

max_fall_delay
: A floating point value that specifies the maximum falling delay on ports,

clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

max_metal_layer

Specifies the reserved maximum metal layer name of a soft macro or a black box.
This attribute is read-only and cannot be modified.

max_rise_delay

A floating point value that specifies the maximum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

max_time_borrow

A floating point number that establishes an upper limit for time borrowing; that is, it prevents the use of the entire pulse width for level-sensitive latches. Units are those used in the technology library. Set with **set_max_time_borrow**.

min_fall_delay

A floating point value that specifies the minimum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

min_metal_layer

Specifies the reserved minimum metal layer name of a soft macro or a black box.
This attribute is read-only and cannot be modified.

min_rise_delay

A floating point value that specifies the minimum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

minus_uncertainty

Specifies a negative uncertainty from the edges of the ideal clock waveform. Affects all clock pins on this cell. Set with **set_clock_skew -minus_uncertainty**.

number_of_black_box

Specifies the count of black boxes in a hierarchical cell or a soft macro. Whether a cell is a black box can be determined by the attribute is_black_box.
This attribute is read-only and cannot be modified.

number_of_io_cell

Specifies the count of io cells in a hierarchical cell or a soft macro. The tool counts cells in when its mask_layout_type is **io_pad**, **corner_pad**, **pad_filler**, or **flip_chip_pad**.
This attribute is read-only and cannot be modified.

number_of_macro

Specifies the count of macros in a hierarchical cell or a soft macro. The tool counts cells in when its mask_layout_type is macro.
This attribute is read-only and cannot be modified.

number_of_pinshape
: Specifies the number of pin shapes of a soft macro.
This attribute is read-only and cannot be modified.

number_of_standard_cell
: Specifies the count of standard cells in a hierarchical cell or a soft macro.
The tool counts cells in when its **mask_layout_type** matches ***std***.
This attribute is read-only and cannot be modified.

physical_area
: Specifies the physical area of a hierarchical cell or a soft macro.
The physical area of a hierarchical cell is the sum of its direct children's **physical_area** or **area**. If a direct child is a hierarchical cell, then **physical_area** is used. If a child is a standard cell or macro, then **area** is used, otherwise that child is skipped.
The physical area of a soft macro is the sum of its children's **physical_area** or **area**. The children will be iterated from the subdesign file. If a child is a soft macro, then **physical_area** is used, if a child is a standard cell or hard macro, **area** is used, otherwise that child is skipped.
This attribute is read-only and cannot be modified.

physical_area_percentage_in_top_design
: Specifies the percentage of **physical_area** of a soft macro in the top design.
This attribute is read-only and cannot be modified.

plus_uncertainty
: Specifies a positive uncertainty from the edges of the ideal clock waveform. Affects all clock pins on this cell. Set with **set_clock_skew -plus_uncertainty**.

propagated_clock
: Specifies that the clock edge times be delayed by propagating the values through the clock network. Affects all clock pins on this cell. If this attribute is not present, ideal clocking is assumed. Set with **set_clock_skew -propagated**.

ref_name
: The reference name of a cell.
This attribute is read-only and cannot be modified.

rise_delay
: Specifies an offset from the rising edge of the ideal clock waveform. Affects all clock pins on this cell. Set with **set_clock_skew -rise_delay**.

scan
: When *true*, specifies that the cell is always replaced by an equivalent scan cell during **insert_dft**. When *false*, the cell is not replaced. Set with **set_scan**.

scan_chain
: Includes the specified cells of the referenced design in the scan-chain whose index is the value of this attribute. Set with **set_scan_chain**.

test_dont_fault
: Specifies cells not faulted during test pattern generation. If no command

options are specified, this attribute is set for both "stuck-at-0" and "stuck-at-1" faults. Set with **set_test_dont_fault**.

test_isolate
: Indicates that the specified sequential cells, pins, or ports are to be logically isolated and considered untestable during test design rule checking by **check_test**. When this attribute is set on a cell, it is also placed on all pins of that cell. Do not set this attribute on a hierarchical cell. Use **report_test -assertions** for a report on isolated objects. Set with **set_test_isolate**.
Note: Setting this attribute suppresses the warning messages associated with the isolated objects.

test_routing_position
: Specifies the preferred routing order of the scan-test signals of the identified cells. Set with **set_test_routing_order**.

ungroup
: Removes a level of hierarchy by exploding the contents of the specified cell in the current design. If specified on a reference object, cells using that reference are ungrouped during **compile**. Set with **set_ungroup**.

utilization
: Specifies the utilization of a soft macro.
The tool calculates the utilization using **physical_area**:**area** ratio of a soft macro.
This attribute is read-only and cannot be modified.

width
: Specifies the width of a cell. This attribute does not exist on a hierarchical cell.
The tool uses the cell’s cell boundary to calculate its width.
This attribute is read-only and cannot be modified.

SEE ALSO

get_attribute(2)
remove_attribute(2)
set_attribute(2)
attributes(3)

cell_site_attributes

Contains attributes related to cell site.

DESCRIPTION

Contains attributes related to cell site.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified object. Specified with **list_attribute -class cell_site -application**, the definition of attributes can be listed.

Cell Site Attributes

bbox

Specifies the bounding-box of a cell site. The **bbox** is represented by a **rectangle**.
The format of a *rectangle* specification is {{llx lly} {urx ury}}, which specifies the lower-left and upper-right corners of the rectangle.
The data type of **bbox** is string.
This attribute is read-only.

constraints

Specifies the maximum number of flip chip driver of the particular personality type that can be placed in a flip chip driver island.
The data type of **constraints** is string.
This attribute is read-only.

layer

Specifies system layer name on which cell site is.
The data type of **layer** is string.
This attribute is read-only.

object_class

Specifies object class name of a cell site, which is **cell_site**.
The data type of **object_class** is string.
This attribute is read-only.

orientations

Specifies allowable orientations of flip chip driver when placed.
The data type of **orientations** is string.
This attribute is read-only.

personality

Specifies personality types of flip-chip driver cells to be placed in a cell site.
The data type of **personality** is string.
This attribute is read-only.

reserved_slots

Specifies that certain locations pointed by the row and column indices in an flip chip driver island are reserved for a certain flip chip driver cell.

Only the specified flip chip driver can be placed in the locations.
Its format is like, {{**lib_cell_name** {**col row**}...} ...}
The data type of **reserved_slots** is string.
This attribute is read-only.

rotate_by_row

Specifies forced orientations of the drivers are set by rows.
The data type of **rotate_by_row** is boolean.
This attribute is read-only.

style

Specifies the style to place flip chip drivers.
Its valid values can be:

- **flip_chip_array**
- **flip_chip_island**
- **flip_chip_ring**
- **flip_chip_row**

The data type of **style** is string.
This attribute is read-only.

SEE ALSO

get_attribute(2),
list_attribute(2),
report_attribute(2),
set_attribute(2),
set_flip_chip_driver_array(2),
set_flip_chip_driver_island(2),
set_flip_chip_driver_ring(2).

change_names_bit_blast_negative_index

Bit blast the bus if any bit of it is negative.

TYPE

Boolean

DEFAULT

false

GROUP

DESCRIPTION

If this varialbe is set true, change_names will bit blast the bus if any bit is negative. Otherwise, change_names will shift negative range to the positive range starting 0. The default value is *false*.

To determine the current value of this variable, use **printvar change_names_bit_blast_negative_index**.

SEE ALSO

change_names(2)
define_name_rules(2)

change_names_dont_change_bus_members

Controls how the **change_names** command modifies the names of bus members.

TYPE

Boolean

DEFAULT

false

GROUP

system_variables

DESCRIPTION

This variable is for the **change_names** command, and affects bus members only of bussed ports or nets. When false (the default), **change_names** gives bus members the base name from their owning bus. For example, if BUS A has range 0 to 1 with the first element NET1 and the second element NET2, **change_names** changes NET1 to A[0] and NET2 to A[1]. When this variable is set to true, **change_names** does not change the names of bus members, so that NET1 and NET2 remain unchanged.

This variable also applies to **-special** rules, but has no effect if the name is changed by other rules that have higher priority than **-special** when **-special** rules are used. (For example, **-equal_ports_nets**, **-case_insensitive**.) For more information, refer to the **APPLYING NAME RULES** section of the **define_name_rules** manual page.

To determine the current value of this variable, type **printvar change_names_dont_change_bus_members**. For a list of all **system** variables and their current values, type **print_variable_group system**.

SEE ALSO

change_names(2)
define_name_rules(2)
system_variables(3)

check_design_allow_non_tri_drivers_on_tri_bus

Specifies the severity level to be applied during compile or check_design command execution, when three-state buses with non three-state driver(s) are found in the design.

TYPE

Boolean

DEFAULT

true

GROUP

none

DESCRIPTION

When true, **compile** and **check_design** will warn out when three-state buses with non three-state driver(s) exist in the design. These commands will continue execution upon warning. When false, they will report errors on three-state bus. **compile** will cease to continue execution upon encoutering errors and return 0 status. **check_design** will however continue execution without any change in return status.

The default value of this variable is true.

To determine the current value of this variable use **printvar check_design_allow_non_tri_drivers_on_tri_bus**.

SEE ALSO

check_design(2)
compile(2)

check_design_allow_unknown_wired_logic_type

Specifies the severity level to be applied during compile or check_design command execution, when nets with multiple drivers(unknown wired-logic type) are found in the design.

TYPE

Boolean

DEFAULT

true

GROUP

none

DESCRIPTION

When true, **compile** and **check_design** will warn out when nets with multiple drivers(unknown wired-logic type) exist in the design. These commands will continue execution upon warning. When false, they will report errors on such nets. **compile** will cease to continue execution upon encoutering errors and return 0 status. **check_design** will however continue execution without any change in return status.

The default value of this variable is true.

To determine the current value of this variable use **printvar check_design_allow_unknown_wired_logic_type**.

SEE ALSO

check_design(2)
compile(2)

check_error_list

Specifies the error codes that the **check_error** command checks for.

TYPE

list

DEFAULT

CMD-004 CMD-006 CMD-007 CMD-008 CMD-009 CMD-010 CMD-011 CMD-012 CMD-014 CMD-015 CMD-016 CMD-019 CMD-026 CMD-031 CMD-037 DB-1 DCSH-11 DES-001 ACS-193 FILE-1 FILE-2 FILE-3 FILE-4 LINK-7 LINT-7 LINT-20 LNK-023 OPT-100 OPT-101 OPT-102 OPT-114 OPT-124 OPT-127 OPT-128 OPT-155 OPT-157 OPT-181 OPT-462 UI-11 UI-14 UI-15 UI-16 UI-17 UI-19 UI-20 UI-21 UI-22 UI-23 UI-40 UI-41 UID-4 UID-6 UID-7 UID-8 UID-9 UID-13 UID-14 UID-15 UID-19 UID-20 UID-25 UID-27 UID-28 UID-29 UID-30 UID-32 UID-58 UID-87 UID-103 UID-109 UID-270 UID-272 UID-403 UID-440 UID-444 UIO-2 UIO-3 UIO-4 UIO-25 UIO-65 UIO-66 UIO-75 UIO-94 UIO-95 EQN-6 EQN-11 EQN-15 EQN-16 EQN-18 EQN-20

GROUP

acs_variables

DESCRIPTION

Specifies the error codes that the **check_error** command checks for. The **check_error** command returns a 1 if any of the specified error codes have been generated by a previous command in the current dc_shell session.

Automated Chip Synthesis uses this capability to stop batch jobs in which the specified error codes occur.

To determine the current value of this variable, use **printvar check_error_list** in DB (dcsh) mode and **printvar check_error_list** in DB (Tcl) mode.

SEE ALSO

check_error(2)

clock_attributes

Contains attributes placed on clocks.

DESCRIPTION

Contains attributes that can be placed on clocks.

To set an attribute, use the command identified in the individual description of that attribute. To determine the value of an attribute, use the **get_attribute** command. To remove attributes, use the **remove_attribute** command.

For a more detailed explanation of an attribute, refer to the manual pages of the appropriate **set** command. For information on all attributes, refer to the **attributes** manual page.

Clock Attributes

dont_touch_network
: When a design is optimized, **compile** assigns **dont_touch** attributes to all cells and nets in the transitive fanout of **dont_touch_network** ports. The **dont_touch** assignment stops at the boundary of storage elements. An element is recognized as storage only if it has setup or hold constraints. Set with **set_dont_touch_network**.

fall_delay
: Specifies an offset from the falling edge of the ideal clock waveform. Set with **set_clock_skew -fall_delay**

fix_hold
: Specifies that **compile** should attempt to fix hold violations for timing endpoints related to this clock. Set with **set_fix_hold**.

max_fall_delay
: A floating point value that specifies the maximum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

max_rise_delay
: A floating point value that specifies the maximum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

max_time_borrow
: A floating point number that establishes an upper limit for time borrowing; that is, it prevents the use of the entire pulse width for level-sensitive latches. Units are those used in the technology library. Set with **set_max_time_borrow**.

min_fall_delay
: A floating point value that specifies the minimum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

min_rise_delay
: A floating point value that specifies the minimum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

minus_uncertainty
: Specifies a negative uncertainty from the edges of the ideal clock waveform. Set with **set_clock_skew -minus_uncertainty**.

period
: Assigns a value to the clock period. The clock period (or cycle time) is the shortest time during which the clock waveform repeats. For a simple waveform with one rising and one falling edge, the period is the difference between successive rising edges. Set with **create_clock -period period_value**.

plus_uncertainty
: Specifies a positive uncertainty from the edges of the ideal clock waveform. Affects all sequential cells in the transitive fanout of this port. Set with **set_clock_skew -plus_uncertainty**.

propagated_clock
: Specifies that the clock edge times be delayed by propagating the values through the clock network. If this attribute is not present, ideal clocking is assumed. Set with **set_clock_skew -propagated**.

rise_delay
: Specifies an offset from the rising edge of the ideal clock waveform. Set with **set_clock_skew -rise_delay**.

SEE ALSO

get_attribute(2)
remove_attribute(2)
attributes(3)

collection_result_display_limit

Sets the maximum number of objects that can be displayed by any command that displays a collection.

TYPE

int

DEFAULT

100

DESCRIPTION

This variable sets the maximum number of objects that can be displayed by any command that displays a collection. The default is 100.

When a command (for example, **add_to_collection**) is issued at the command prompt, its result is implicitly queried, as though **query_objects** had been called. You can limit the number of objects displayed by setting this variable to an appropriate integer. A value of -1 displays all objects; a value of 0 displays the collection handle id instead of the names of any objects in the collection.

To determine the current value of this variable, use **printvar collection_result_display_limit**.

SEE ALSO

collections(2)
printvar(2)
query_objects(2)

command_log_file

Specifies the name of the file to which a log of the initial values of variables and commands executed is written. If the value is an empty string, a command log file is not created.

TYPE

string

DEFAULT

"./command.log"

GROUP

system_variables

DESCRIPTION

Specifies the name of the file to which a log of the initial values of variables and commands executed is written. If the value is an empty string, a command log file is not created. Also, if "-no_log" has been specified when invoking DC, then command log file is not created.

This variable is only valid in eqn mode of dc_shell. If you are running tcl mode, please use the variable sh_command_log_file.

To determine the current value of this variable use **printvar command_log_file**. For a list of all **system** variables and their current values, use the **print_variable_group system** command.

SEE ALSO

sh_command_log_file(3)
view_command_log_file(3)
view_log_file(3)

company

Specifies the name of the company where Synopsys software is installed. The company name is displayed on the schematics.

TYPE

string

DEFAULT

""

GROUP

system_variables

DESCRIPTION

Specifies the name of the company where Synopsys software is installed. The company name is displayed on the schematics.

To determine the current value of this variable use **printvar company**. For a list of all **system** variables and their current values, use the **print_variable_group system** command.

compatibility_version

Sets the default behavior of the system to be the same as the Synopsys software version specified in the variable.

TYPE

string

DEFAULT

C-2009.06

GROUP

system_variables

DESCRIPTION

Sets the default behavior of the system to be the same as the Synopsys software version specified in the variable. This setting provides compatibility for script command files written in previous software versions. The scripts are run on the current version of the software, so results are usually better. However, the script performs the same default actions here as it did on the specified software version.

To determine the current value of this variable use **printvar compatibility_version**. For a list of all **system** variables and their current values, use the **print_variable_group system** command.

compile_allow_dw_hierarchical_inverter_opt

Allows the phase inversion boundary optimization to be applied to DesignWare components.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

The **compile_allow_dw_hierarchical_inverter_opt** variable allows the phase inversion boundary optimization to be applied to DesignWare components, when boundary optimization in general is permitted on the component.

The default value of this variable is **false**, which will prevent the compile command from changing the phase of DesignWare component output signals. Other boundary_optimizations will still be attempted. Setting the variable to **true** will enable the phase inversion on DesignWare component outputs; note that the **true** setting may interfere with formal verification of the design.

SEE ALSO

compile(2)

compile_assume_fully_decoded_three_state_busses

Specifies whether the **compile** and **translate** commands can assume that three-state busses are fully decoded.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

When true, **compile** and **translate** assume that three-state busses are fully decoded and therefore can be replaced by multiplexed busses when mapping to a library that contains no three-state cells. Default is false.

To determine the current value of this variable use **printvar compile_assume_fully_decoded_three_state_busses**. For a list of **compile** variables and their current values, use the **print_variable_group compile** command.

SEE ALSO

compile_variables(3)

compile_auto_ungroup_area_num_cells

Defines the minimum number of child cells a design hierarchy must have so that the command **compile -auto_ungroup area** does not ungroup the hierarchy.

TYPE

integer

DEFAULT

30

GROUP

compile_variables

DESCRIPTION

The **compile_auto_ungroup_area_num_cells** variable defines the minimum number of child cells a design hierarchy must have so that the **compile -auto_ungroup area** command does not ungroup the hierarchy. The default value for this variable is 30. By default the threshold check is done only on the child cells of its immediate hierarchy. Users, however, could enable the threshold check to include the all the leaf cells of this design hierachy, i.e., both child cells of its immediate hierarchy and all its sub designs by setting the variable **compile_auto_ungroup_count_leaf_cells** to true.

To determine the current value of this variable, type **printvar compile_auto_ungroup_area_num_cells**. For a list of all **compile** variables and their current values, type **print_variable_group compile**.

SEE ALSO

compile(2)
compile_auto_ungroup_count_leaf_cells(3)
compile_auto_ungroup_override_wlm(3)

compile_auto_ungroup_count_leaf_cells

Determines if the number of leaf cells should be counted or the number of child cells in the immediate hierarchy should be counted to compare against the value of variable **compile_auto_ungroup_area_num_cells** in area-based auto-ungroup

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

The setting of **compile_auto_ungroup_count_leaf_cells** variable determines the way we count the number of child cells of a design hierarchy. When this variable is set to false, which is its default value, only child cells in the immediate hierarchy are counted, the number of child cells in its subdesigns is not taken into consideration. When this variable is set to true, the leaf cells of a design hierarchy, i.e., both child cells of its immediate hierarchy and all its subdesigns are taken into consideration. The number of child cells of a design hierarchy is compared against the value of **compile_auto_ungroup_area_num_cells** in area-based auto-ungrouping to determine if a design hierarchy can be considered as a candidate for ungrouping.

Because of this, for the same design hierarchy and same variable settings for auto-ungroup, a certain design hierarchy may be ungrouped if **compile_auto_ungroup_count_leaf_cells** is set to false, but may not be ungrouped if it is set to true. For example, let's say design hierarchy A has 20 immediate child cells and a subdesign B and subdesign B has another 40 immediate child cells with no more subdesigns. In a compile flow where **compile -auto_ungroup area** is used and variable **compile_auto_ungroup_area_num_cells** is set to 30, if **compile_auto_ungroup_count_leaf_cells** is set to false, design hierarchy A will be considered as a candidate for auto-ungrouping because it has 20 < 40 child cells in its immediate hierarchy. However, if **compile_auto_ungroup_count_leaf_cells** is set to true, since design A has 20+40 = 60 > 40 leaf cells, it will be not be considered as a candidate for auto-ungrouping.

To determine the current value of this variable, type **printvar compile_auto_ungroup_count_leaf_cells**. For a list of all **compile** variables and their current values, type **print_variable_group compile**.

SEE ALSO

compile(2)

compile_auto_ungroup_area_num_cells(3)

compile_auto_ungroup_override_wlm

Specifies whether the compiler considers a cell instance for automatic ungrouping, if the cell’s wire load model differs from that of its parent.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

Specifies whether the compiler considers a cell instance for automatic ungrouping, if the cell’s wire load model is different from the wire load model of its parent cell. The default value is *false*, which means that the cell instance is not considered for automatic ungrouping should its wire load model differ from that of its parent.

If you set the value of this variable as *true*, the ungrouped child cells of the named cell instance inherits the wire load model of its parent. A result of setting this variable to *true* is that the tool might use more pessimistic wire load models for the child cells of this cell instance. This in turn might offset the delay improvement from **compile -auto_ungroup area|delay** and lead to seemingly worse timing behavior for the design.

To determine the current value of this variable, type **printvar compile_auto_ungroup_override_wlm**. For a list of all **compile** variables and their current values, type **print_variable_group compile**.

SEE ALSO

compile(2)
compile_auto_ungroup_area_num_cells(3)

compile_automatic_clock_phase_inference

Specifies the method used to determine clock phase during sequential mapping.

TYPE

string

DEFAULT

strict

GROUP

compile_variables

DESCRIPTION

When set to *strict*, **compile** will attempt to determine the desired clock phase for each unmapped register, and will not allow opposite phase devices to be used in constructing registers. When set to *relaxed*, **compile** will allow the implementation of an opposite phase device for a register only if there is no other way to implement that register. When set to *none*, **compile** will ignore clock phase during sequential mapping. The default is *strict*.

To determine the current value of this variable use **printvar compile_automatic_clock_phase_inference**. For a list of **compile** variables and their current values, use the **print_variable_group compile** command.

SEE ALSO

compile_variables(3)

compile_checkpoint_phases

Determines whether checkpoints are generated during execution of the **compile** command.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

When true, checkpoints automatically between each phase of **compile**. The default is false.

To determine the current value of this variable, type **printvar compile_checkpoint_phases**. For a list of all **compile** variables and their current values, type **print_variable_group compile**.

SEE ALSO

compile_clock_gating_through_hierarchy

Controls whether the **compile** or **compile_ultra** command with the **-gate_clock** option will perform clock gating through hierarchy boundaries.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

When true, **compile -gate_clock** and **compile_ultra -gate_clock** are allowed to use one clock gate to gate registers in different hierarchical cells. This can increase the number of clock gating opportunities and reduce the number of clock gates.

When false (the default), the clock gating will only be performed in such a way that clock gates are in the same hierarchy cell as all the registers gated by them.

To determine the current value of this variable use **printvar compile_clock_gating_through_hierarchy**. For a list of all **compile** variables and their current values, use the **print_variable_group compile** command.

SEE ALSO

compile_variables(3)

compile_cpu_limit

Specifies a time, in minutes, to be used as the limit for the amount of time to be spent in the phases after structuring and mapping. Optimization cancels when the limit is reached.

TYPE

float

DEFAULT

0.0

GROUP

compile_variables

DESCRIPTION

Specifies a time, in minutes, to be used as the limit for the amount of time to be spent in the phases after structuring and mapping. Optimization cancels when the limit is reached. The default value, "0.0", indicates that there is no limit.

To determine the current value of this variable, use **printvar compile_cpu_limit**. For a list of all **compile** variables and their current values, use the **print_variable_group compile** command.

SEE ALSO

compile (2).

compile_create_wire_load_table

Controls the type of wire load model generated by the **create_wire_load** command.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables
links_to_layout_variables

DESCRIPTION

This variable is used to control the type of wire load model generated by the **create_wire_load** command. The default setting of this variable is false and the wire load models generated are in the wire_load format. It includes resistance, capacitance, area slope coefficients, and fanout_length (fanout, length, average_cap, std_dev, and points)

If this variable is set to true the wire load models generated are in the wire_load_table format. The resistance is calculated explicitly for each fanout using the back-annotated values and the tree type from the current operating condition set on the design.

To determine the current value of this variable, use **printvar compile_create_wire_load_table**.

For a list of all **compile** variables and their current values, use the **print_variable_group compile** command. For a list of all **links_to_layout** variables and their current values, type **print_variable_group links_to_layout**.

SEE ALSO

compile (2), **compile_variables** (3), **links_to_layout_variables** (3).

compile_delete_unloaded_sequential_cells

Controls whether the **compile/physopt/compile_physical** command deletes unloaded sequential cells.

TYPE

Boolean

DEFAULT

true

GROUP

compile_variables

DESCRIPTION

A design can contain sequential cells that drive no loads. During **compile/physopt/compile_physical**, the logic driven by a sequential cell might be optimized away, resulting in an inferred no-load, or no path to any primary output. By default, **compile/physopt/compile_physical** deletes such sequential cells. To retain such cells, set variable **compile_delete_unloaded_sequential_cells** to false.

SEE ALSO

compile(2)
compile_variables(3)

compile_disable_hierarchical_inverter_opt

Controls whether inverters can be moved across hierarchical boundaries during boundary optimization.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

This variable affects the behaviour of the boundary-optimization feature in the **compile** command. By default, boundary-optimization will try to push inverters across hierarchy to improve the optimization cost of the design. However, if the **compile_disable_hierarchical_inverter_opt** variable is set to true, boundary-optimization will not move inverters across hierarchical boundaries even if that could have improved the design.

To determine the current value of this variable use **printvar compile_disable_hierarchical_inverter_opt**. For a list of all **compile** variables and their current values, use the **print_variable_group compile** command.

SEE ALSO

compile(2)
set_boundary_optimization(2)
port_complement_naming_style(3)
compile_variables(3)

compile_dont_touch_annotated_cell_during_inplace_opt

Controls whether cells that have annotated delays can be optimized.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables
links_to_layout_variables

DESCRIPTION

When true, **reoptimize_design -in_place** and **compile -in_place** disallow swapping cells that have annotated delays. When false (the default value), **reoptimize_design -in_place** and **compile -in_place** allow annotated cells to be swapped for cells without annotated delay.

This variable is used to run **reoptimize_design -in_place** and **compile -in_place** with annotated cell and net delays and allowing only buffers to be inserted. When this variable is true, Design Compiler considers all cells with annotated delays as **dont_touch**. This allows avoiding exchanging a cell with annotated delays for a cell with lower estimated delays but higher real delays.

To determine the current value of this variable, use **printvar compile_dont_touch_annotated_cell_during_inplace_opt**. For a list of all **compile** variables and their current values, use the **print_variable_group compile** command. For a list of all **links_to_layout** variables and their current values, use the **print_variable_group links_to_layout** command.

SEE ALSO

compile(2), compile_variables(3), links_to_layout_variables (3).

compile_dont_use_dedicated_scanout

Controls whether optimizations use a scan cell’s dedicated scan-out pin for functional connections.

TYPE

integer

DEFAULT

1

GROUP

insert_dft_variables

DESCRIPTION

When 1 (the default), optimizations (**place_opt, clock_opt, route_opt and psynopt**) do not use a scan cell’s dedicated scan-out pin for functional connections.

When 0, optimizations can use dedicated scan-out pins for functional connections.

Dedicated scan-out pins must be identified in the technology library using the **test_output_only** attribute. Contact your ASIC Vendor to ensure that dedicated scan-out pins are correctly modeled in the library that you are using.

To determine the current value of this variable, type **printvar compile_dont_use_dedicated_scanout**.

SEE ALSO

compile_enable_dyn_max_cap

Determines whether the tool is to use the operating frequency to determine the maximal pin load.

TYPE

Boolean

DEFAULT

false

GROUP

timing_variables

DESCRIPTION

The value you specify for the **compile_enable_dyn_max_cap** variable determines whether the tool uses a single value for the maximal pin load (default) or uses the operating frequency to determine the maximal pin load. The default value of this variable is **false**.

If you want the tool to use the operating frequency to determine the maximal pin load, set the value of this variable to **true**.

To see the current value of this variable, type

```
dc_shell-xg-t> printvar compile_enable_dyn_max_cap
```

SEE ALSO

compile(2)
set_max_capacitance(2)
set_min_capacitance(2)

compile_fix_cell_degradation

Controls whether the algorithms for fixing cell degradation violation are activated.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

When true, the algorithms for fixing cell degradation violations in **compile** and **reoptimize_design** are activated. Different strategies, such as sizing and buffering, try to fix violations of the cell degradation design rule.

To determine the current value of this variable, type **printvar compile_fix_cell_degradation**. For a list of all compile variables and their current values, type **print_variable_group compile**.

SEE ALSO

compile (2), **compile_variables** (3), **report_constraints** (2).

compile_hold_reduce_cell_count

Controls whether the logic used to fix hold time violations is selected based on minimum cell count or minimum area.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

When true, Design Compiler uses the minimum number of cells to fix the hold time (min path) violations, rather than choosing cells that minimize the total new area. This means that the area may be worsened (compared to the default flow) while the hold time violations are being fixed.

SEE ALSO

compile_variables (3)

compile_implementation_selection

Controls whether the **compile** command reevaluates the current implementation of a synthetic module during optimization.

TYPE

Boolean

DEFAULT

true

GROUP

compile_variables

DESCRIPTION

When true (the default), **compile** re-evaluates the current implementation of a synthetic library module and replaces it if appropriate for optimizing the design. When false, this optimization is disabled, saving CPU time with a potential loss of quality of results.

To determine the current value of this variable, type **printvar compile_implementation_selection**. For a list of **compile** variables and their current values, type **print_variable_group compile**.

SEE ALSO

compile(2)
set_implementation(2)
compile_variables(3)

compile_instance_name_prefix

Specifies the prefix used in generating cell instance names when **compile** is executed.

TYPE

string

DEFAULT

U

GROUP

compile_variables

DESCRIPTION

Specifies the prefix used in generating cell instance names when **compile** is executed.

To determine the current value of this variable use **printvar compile_instance_name_prefix**. For a list of **compile** variables and their current values, use the **print_variable_group compile** command.

SEE ALSO

compile_variables(3)

compile_instance_name_suffix

Specifies the suffix used for generating cell instance names when **compile** is executed.

TYPE

string

DEFAULT

""

GROUP

compile_variables

DESCRIPTION

Specifies the suffix used for generating cell instance names when **compile** is executed.

To determine the current value of this variable use **printvar compile_instance_name_suffix**. For a list of **compile** variables and their current values, use the **print_variable_group compile** command.

SEE ALSO

compile_variables(3)

compile_keep_original_for_external_references

Controls compile command to keep the original design when there is an external reference to the design.

TYPE

Boolean

DEFAULT

false

GROUP

none

DESCRIPTION

By default, **compile** will modify the original copy of the designs in the current design.

When set to true, the original design and its sub designs are copied and preserved (before doing any modifications during compile) if there is an external reference to this design.

For example, if there is an instance of design bot, U1 in the current design mid and there is an external reference from another design top which is not part of current hierarchy, then the U1 will be uniquified to a new design bot_0 before doing any modification to the design. Hence when the user changes the current_design to top and performs a link, the original design bot will be linked into the current_hierarchy.

Usually, this is needed only when you are doing a bottom compile without setting dont_touches on all the sub designs especially with boundary optimization turned on during compile.

If there is a dont_touch attribute on any of the instances of the design or in the design itself, this variable does not have effect.

SEE ALSO

compile(2)

compile_log_format

Controls the format of the columns to be displayed during the mapping phases of **compile** and **reoptimize_design**.

TYPE

string

DEFAULT

%elap_time %area %wns %tns %drc %endpoint

GROUP

compile_variables

DESCRIPTION

Controls the format of the columns to be displayed during the mapping phases of **compile** and **reoptimize_design**. The default specification is shown in the DEFAULT section and results in an output display format similar to Table 1. The headings and order of the columns displayed correspond to the keywords specified in the syntax. For example, "%elap_time" specifies the ELAPSED TIME column, "%area" the AREA column, and so on.

Table 1
Default compile Log Output Format

```
 Elapsed              Worst Neg    Total Neg        Design

  Time       Area       Slack          Slack         Rule Cost          Endpoint

---------       --------- --------   -----------     ---------       ----------------
----------
 18:00:30     1498.0       4.12            32.2        0.0        U1/U2/
CURRENT_SECS_reg[4]
 18:00:30     1498.0       4.07            31.6        0.0        U1/U2/
CURRENT_SECS_reg[4]
 18:00:30     1497.0       4.07            30.6        0.0        U1/U2/
CURRENT_SECS_reg[4]
 18:00:31     1499.0       3.61            27.5        0.0        U1/U2/
CURRENT_SECS_reg[4]
 18:00:31     1499.0       3.58            26.1        0.0        U1/U2/
CURRENT_SECS_reg[4]
```

By default, the columns in Table 1 are nine characters wide except for the ENDPOINT column, which is 25 characters. The default precision for the floating point data types AREA, TOTAL NET SLACK, and DESIGN RULE COST is one digit to the right of the decimal point; for WORST NEG SLACK, two digits.

There are 13 possible columns that can be displayed; only six are displayed in the default format. You can create a customized output format by specifying any number of the available columns, with their keywords and defaults. For example, specifying "%mem" displays the MBYTES column with a width of 6 characters and a precision of 1 digit to the right of the decimal point. When you specify "%mem", the following information is displayed horizontally under these column names: COLUMN HEADER, DATA TYPE, KEYWORD, WIDTH, PRECISION, and FORMAT.

MBYTES floating point mem 6 1 f

See the section "DEFINITIONS OF COLUMN FIELDS WITH DEFAULT VALUES" for descriptions and contents of the fields corresponding to the column headers.

CHANGING DEFAULT COLUMN PARAMETERS

You can change the default column parameters using the optional expression (w.pf,split), as follows:

compile_log_format = " %elap_time %area %wns %tns %drc %endpoint".

string **compile_log_format** = "%keyword(w.pf,split)"

The quantities w, p, f, and split are defined as follows.

w

Specifies the column width. Specifying a width less than 6 defaults the width to 6. For string data types, specifying a width greater than 99 defaults the width to 99; for decimal or floating point data types, specifying a width greater than 25 defaults to 25.

p

For floating point numbers only. Specifies the precision in number of digits. See also the definition of f.

f

For floating point numbers only. Specifies the precision format; values are *f* or *g*. *f* specifies that the precision is expressed as the number of digits to the right of the decimal point; *g* specifies that the precision is expressed as the total number of significant digits. For example, expressing the floating point number 13.533 with a precision of 3 in format *f* reports the number as 13.533; in format *g*, 13.5.

split

By default, if the information in a given field exceeds the column width, it pushes out the next field and the next field is not printed on a new line. Specifying **split** overrides the default and causes the next field to begin on a new line, starting in the correct column.
%elap_time %area %wns %tns %drc %endpoint

1998.02 COMPILE LOG FORMAT

The fields for the 1998.02 version are TRIALS, AREA, DELTA DELAY, TOTAL NEGATIVE SLACK, and DESIGN RULE COST. The DELTA DELAY field is renamed MAX DELAY COST in the 1998.08 format.

To display the same log format as the 1998.02 version, set the variable as follows:

compile_log_format = ""

DEFINITIONS OF COLUMN FIELDS WITH DEFAULT VALUES

The fields listed under the five columns are defined in the following text, showing the default values. Except where noted, units are those defined by the library.

AREA
    Shows the area of the design during the optimization.
    Data type: floating point
    Keyword: are
    Width: 9
    Precision: 1
    Format: f

CPU SEC
    Shows the process cpu time used, in seconds.
    Data type: decimal
    Keyword: cpu
    Width: 7
    Precision: ignored
    Format: ignored

DELTA DELAY
    See MAX DELAY COST.

DESIGN RULE COST
    Measures the distance between the actual results and user-specified design rule constraints.
    Data type: floating point
    Keyword: drc
    Width: 9
    Precision: 1
    Format: f

ELAPSED TIME
    Tracks the elapsed time since the beginning of the current **compile** or **reoptimize_design**.
    Data type: string
    Keyword: elap_time
    Width: 9
    Precision: ignored
    Format: ignored

MAX DELAY COST
    Shows the current max delay cost of the design, which is the sum of the worst

```
          negative slack (max_path violation) in each path group. Called "DELTA DELAY"
          in the 1998.02 version.
          Data type: floating point
          Keyword: max_delay
          Width: 9
          Precision: 2
          Format: f

MBYTES
          Shows the process memory used, in mbytes.
          Data type: floating point
          Keyword: mem
          Width: 6
          Precision: 1
          Format: f

MIN DELAY COST
          Shows the current min delay cost of the design, which is the sum of the worst
          negative slack (min_path violation) in each path group.
          Data type: floating point
          Keyword: min_delay
          Width: 9
          Precision: 2
          Format: f

TIME OF DAY
          Shows the current time.
          Data type: string
          Keyword: time
          Width: 8
          Precision: ignored
          Format: ignored

TOTAL NEG SLACK
          Shows the sum of the negative slack across all endpoints in the design.
          Data type: floating point
          Keyword: tns
          Width: 9
          Precision: 1
          Format: f

TRIALS
          Tracks the number of transformations that the optimizer tries before making
          the current selection.
          Data type: decimal
          Keyword: trials
          Width: 6
          Precision: ignored
          Format: ignored

WORST NEG SLACK
          Shows the worst negative slack (max_path violation) in all path groups.
          Data type: floating point
          Keyword: wns
          Width: 9
```

```
        Precision: 2
        Format: f

ENDPOINT
        Shows the current endpoint being worked on. When the delay violation is being
        fixed, the object for the ENDPOINT is a cell or a port. When the design rule
        violations are being fixed, the object for the ENDPOINT is a net.
        Data type: string
        Keyword: endpoint
        Width: 25
        Precision: ignored
        Format: ignored

PATH GROUP
        Shows the current path group of a valid endpoint.
        Data type: string
        Keyword: group_path
        Width: 10
        Precision: ignored
        Format: ignored

DYNAMIC POWER
        Shows the dynamic power of the design during optimization.
        Data type: floating point
        Keyword: dynamic_power
        Width: 9
        Precision: 4
        Format: f

LEAKAGE POWER
        Shows the leakage power of the design during optimization.
        Data type: floating point
        Keyword: leakage_power
        Width: 9
        Precision: 4
        Format: f

TOTAL POWER
        Shows the total power of the design during optimization. (total_power =
        dynamic_power + leakage_power)
        Data type: floating point
        Keyword: total_power
        Width: 9
        Precision: 4
        Format: f
        %elap_time %area %wns %tns %drc %endpoint
```

EXAMPLES

The following example increases the precision of the WORST NEG SLACK column by 1 (to 3 digits from its default of 2 digits).

```
prompt> compile_log_format = " %elap_time %area %wns(.3) %tns %drc %endpoint"
```

The following example replaces the TOTAL NEG SLACK column in the default format, with the CPU column.

```
prompt> compile_log_format = " %elap_time %area %wns %cpu %drc %endpoint"
```

In the following example, if the ENDPOINT value exceeds the column width, the next field begins on a new line, starting in the correct column.

```
prompt> compile_log_format = " %elap_time %wns %endpoint(19,split) %group_path"
```

The following example displays the MIN DELAY COST column only, changes the column width to 12 characters from the default of 9, and expresses the value with a precision of 3 significant digits.

```
prompt> compile_log_format = " %min_delay(12.3g)
```

The following example sets the compile log to the same format as the 1998.02 version.

```
prompt> compile_log_format = ""
```

To determine the current value of this variable, type **printvar compile_log_format**. For a list of all **compile** variables and their current values, type **print_variable_group compile**.

SEE ALSO

compile(2)
compile_variables(3)

compile_negative_logic_methodology

Specifies the logic value connected to floating inputs by the **compile** and **translate** commands.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

When true, **compile** and **translate** connect floating inputs to logic 1. The default value (false) causes floating inputs to be connected to logic 0.

This variable assignment should be placed at the beginning of the **.synopsys_dc.setup** file (or dc_shell session) and the value should not be changed during the session.

To determine the current value of this variable use **printvar compile_negative_logic_methodology**. For a list of all **compile** variables and their current values, use the **print_variable_group compile** command.

SEE ALSO

compile_variables(3)

compile_no_new_cells_at_top_level

Controls whether the **compile** command adds new cells to the top-level design.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

When true, no new cells are added to the top-level design of the hierarchy during **compile**. New cells are added only to lower levels.

This variable is used when the original design has no top-level cells, to prevent the addition of new cells when adding buffers for timing optimization and design rule fixes.

If the design has leaf cells at the top level, the cells are optimized as normal, so this variable should be set to false. Setting this variable to true means compile will not add any cells during buffering and design rule fixing, even if the design is flat.

To determine the current value of this variable use **printvar compile_no_new_cells_at_top_level**. For a list of all **compile** variables and their current values, use the **print_variable_group compile** command.

SEE ALSO

compile_variables(3)

compile_power_domain_boundary_optimization

Sets the variable to false to disable boundary optimization across power domain boundaries.

TYPE

Boolean

DEFAULT

true

GROUP

mv

DESCRIPTION

This variable controls whether to allow boundary optimization across all power domain boundaries. By default, boundary optimization across power domain boundaries is enabled in **compile_ultra**, unless specifically disabled by a command such as **set_boundary_optimization**. This variable provides an automatic way to disable boundary optimization across all power domain boundaries.

SEE ALSO

set_boundary_optimization(2)

compile_preserve_subdesign_interfaces

Controls whether the **compile** command preserves the subdesign interface.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

When true, disables customization of logic external to a subdesign during **compile**, and preserves the subdesign interface. When false (the default), **compile** customizes the logic external to a subdesign based on the subdesign's internal logic.

Optimization across hierarchy can take place in two different ways.

1.) Optimizing the logic external to a subdesign, based on that specific subdesign

In this type of optimization, the external logic is customized based on the contents of the subdesign. For example, if one of the output ports of the subdesign is internally grounded, then that constant can be pushed out to the higher level of the design and used at that level to further optimize the design. With this optimization, top-level ("pins-out") functionality of all blocks is preserved. However, if the specification of the lower-level subdesign is then changed; for example, so that the output port is instead tied to a logic one, the higher level of the design cannot be updated accordingly. The logic propagation already occurred based on the original implementation of the subdesign. By default, **compile** performs this type of optimization; setting **compile_preserve_subdesign_interfaces** to true disables the optimization.

2.) Optimizing the logic internal to a subdesign, based on its specific instantiation

In Design Compiler, hierarchical optimization of a subdesign's internal logic is referred to as boundary_optimization. By default, **compile** does not perform boundary_optimization. You can enable boundary_optimization for a particular module or for the entire design by executing **compile -boundary_optimization** or by setting the **boundary_optimization** attribute on the desired object using **set_boundary_optimization**. As a result of boundary optimization, the subdesign is customized for its specific environment. Boundary_optimization results in an improved overall cost, but the subdesign is no longer functionality equivalent to its original specification. Therefore, the subdesign cannot be reused in an environment that does not match the environment for which it was optimized. For example, if one of the subdesign’s input pins is tied externally to a logic zero, boundary optimization can push that logic constant into the subdesign and further optimize the logic in the subdesign. However, if this subdesign is then used in any environment where the input is no longer tied to a logic zero, the resulting design can be logically incorrect. **compile_preserve_subdesign_interfaces** has no effect on objects that are enabled for boundary optimization.

By default, compile performs (1) described above, but not (2). When boundary_optimization is enabled, both types of hierarchical optimization are performed.

Set **compile_preserve_subdesign_interfaces** to true whenever the internal functionality of a subdesign might change in the future; in that case, **compile** should be disabled from customizing the external logic based on the internal logic of the block. When the internal logic of all blocks has finally stabilized, you can then set this variable back to false to allow further optimization of external logic based on the internal logic of the submodules.

To determine the current value of this variable,type **printvar compile_preserve_subdesign_interfaces**. For a list of **compile** variables and their current values, type **print_variable_group compile**.

SEE ALSO

compile(2)
set_boundary_optimization(2)
compile_variables(3)

compile_retime_exception_registers

Controls whether registers with common path exceptions, including max_path, min_path, multicycle_path, false_path, and group_path, can be moved by adaptive retiming.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

This variable controls whether registers with common path exceptions, including max_path, min_path, multicycle_path, false_path, and group_path, can be moved by adaptive retiming.

For example, if an SDC script contains the following command where **reg** is a register, there is a path exception on **reg**:

```
set_max_delay 1.0 -to reg
```

The variable only affects flows that use the **compile_ultra** command with the **-retime** option.

Allowed values are **false** (the default) and **true**.

If **false** is specified, adaptive retiming does not attempt to move registers with exceptions to improve timing or area.

If **true** is specified, adaptive retiming may try to move registers with the max_path, min_path, multicycle_path, false_path, and group_path path exceptions to improve timing or area.

SEE ALSO

compile_ultra(2)

compile_retime_license_behavior

Controls how the compile command behaves when the **optimize_registers** or **balance_registers** attributes are set on a design or parts of a design and the required license(s) (BOA-BRT or DC-Expert) are not available immediately.

TYPE

string

DEFAULT

wait

GROUP

compile_variables

DESCRIPTION

Controls how the compile command behaves when the optimize_registers or balance_registers attributes are set on a design or parts of a design and the required license(s) (BOA-BRT or DC-Expert) are not available immediately.

Allowed values are *wait* (the default), *stop*, or *next*.

If *wait* is selected compile waits in and checks every 5 minutes until the license becomes available. Execution of compile is then resumed.

If *stop* is selected compile is aborted immediately when the required license is not available.

If *next* is selected retiming is executed with the a license allowing less capabilities. I.e. if the optimize_registers attribute is set and the BOA-BRT license is not available, retiming will be executed with the DC-Expert license. QoR will usually be worse. For the balance_registers attribute there is no lower level capability. I.e. *next* has the same effect as *stop* for the balance_registers attribute.

To avoid any waiting or aborting of the compile command because of retiming licensing it is best to obtain the required license in dc_shell before the start of compile. This can be done by using **get_license BOA-BRT** or **get_license DC-Expert** and repeating this until the command is successful.

To determine the current value of this variable, type **printvar compile_retime_license_behavior**. For a list of all **compile** variables and their current values, type **print_variable_group compile**.

SEE ALSO

set_optimize_registers(2)

```
set_balance_registers(2)
```

compile_seqmap_enable_output_inversion

Controls whether the **compile** command allows sequential elements to have their output phase inverted. Has no effect on the **compile_ultra** command.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

When the value of this variable is *true*, the **compile** command allows the mapping of the sequential elements in the design to library cells whose output phase is inverted. This can help improve QoR. It is also useful when mapping sequential cells to a target library whose sequential cells have only one type of asynchronous inputs (either set or reset). In such a case, the only way to match a sequential cell that uses the missing asynchronous input is to use a library cell with the other type of asynchronous input and to invert the output of that cell.

This variable has no effect on the **compile_ultra** command. To control sequential output inversion for **compile_ultra**, use the **-no_seq_output_inversion** option to that command.

Note: When using sequential output inversion, it will be necessary to use the SVF file to verify the functionality of the design using Formality.

To determine the current value of this variable, use **printvar compile_seqmap_enable_output_inversion**. For a list of all **compile** variables and their current values, type **print_variable_group compile**.

SEE ALSO

compile_variables(3)
compile_ultra(2)

compile_seqmap_identify_shift_registers

Controls the identification of shift registers in **compile -scan**. This feature is only supported in test-ready compile with Design Compiler Ultra with a multiplexed scan-style.

TYPE

Boolean

DEFAULT

true

GROUP

compile_variables

DESCRIPTION

When the value of this variable is set to the default value of true, Design Compiler Ultra automatically identifies shift registers in the design during test-ready compile.

When all of the shift registers are identified, only the first register is mapped to a scan cell, while the remaining registers are mapped to non-scan cells. This can save a significant amount of area for designs containing many identified shift registers.

Once these shift registers are identified by Design Compiler Ultra, DFT Compiler will also recognize the identified shift registers as shift-register scan segments. But DFT Compiler will break these scan segments, if necessary, to respect test setup requirements such as maximum chain length.

Shift registers that contain synchronous logic between the registers can also be identified if the synchronous logic can be controlled such that the data can be shifted from the output of the first register to the input of the next register. This synchronous logic can either be internal to the register (for example, synchronous reset and enable) or it can be external synchronous logic (for example, multiplexor logic between the registers). For shift registers identified with synchronous logic between the registers, DFT Compiler will add additional logic to the scan-enable signal during scan insertion in order to allow the data to be shifted between the registers when in scan mode. This capability is controlled by the **compile_seqmap_identify_shift_registers_with_synchronous_logic** variable, and is enabled by default. See the **compile_seqmap_identify_shift_registers_with_synchronous_logic** variable man page for details.

Shift-register identification is only supported in test-ready compile with Design Compiler Ultra with a multiplexed scan-style.

Set the **compile_seqmap_identify_shift_registers** variable to false if you do not want **compile_ultra -scan** to identify shift registers, or if you want to re-scan the shift registers already identified in the design back to scan cells.

The **compile_seqmap_identify_shift_registers_with_synchronous_logic** variable does not have any effect when shift-register identification is disabled with the **compile_seqmap_identify_shift_registers** variable.

SEE ALSO

compile_seqmap_identify_shift_registers_with_synchronous_logic(3)
compile_variables(3)

compile_seqmap_identify_shift_registers_with_synchronous_logic

Controls whether shift registers that contain synchronous logic between the registers are identified. This variable only has an effect in Design Compiler Ultra optimization when shift-register identification is enabled with the **compile_seqmap_identify_shift_registers** variable.

TYPE

Boolean

DEFAULT

true

GROUP

compile_variables

DESCRIPTION

When the value of this variable is set to true and the value of the **compile_seqmap_identify_shift_registers** variable is set to true, during test-ready compile Design Compiler Ultra automatically identifies shift registers that contain synchronous logic between the registers if the synchronous logic can be controlled such that data can be shifted from the output of the first register to the input of the next register. This synchronous logic can either be internal to the register (for example, synchronous reset and enable) or it can be external synchronous logic (for example, multiplexor logic between the registers).

When all of the shift registers are identified, only the first register is mapped to a scan cell, while the remaining registers are mapped to non-scan cells. This can save a significant amount of area for designs containing many identified shift registers. This feature is only available with test-ready compile when using a multiplexed scan style.

Once these shift registers are identified by Design Compiler Ultra, DFT Compiler will also recognize these identified shift registers as shift-register scan segments. But, DFT Compiler will break these scan segments, if necessary, to respect test setup requirements, such as max chain length. For shift registers identified with synchronous logic between the registers, DFT Compiler will add additional logic to the scan-enable signal during scan insertion in order to allow the data to be shifted between the registers. The extra logic results in shared paths between the scan-enable signal and the functional logic so it is important not to set a **dont_touch_network** attribute on the scan-enable ports or signals. A **dont_touch_network** attribute on the scan-enable signal propagates into functional logic paths that prevent optimization of those paths and can lead to QoR degradation.

To disable timing optimization, use the **set_case_analysis** command on scan-enable ports. To disable DRC fixing, use the **set_ideal_network** command on the scan-enable ports. If a **dont_touch_network** attribute must be set, then use **set_dont_touch_network ,Äìno_propagate** instead, to avoid propagation of dont_touch into functional logic.

Set the **compile_seqmap_identify_shift_registers_with_synchronous_logic** variable to false if you do not want Design Compiler Ultra to identify shift registers containing synchronous logic between the registers, or if your design flow does not permit the insertion of additional logic on the scan-enable signal.

SEE ALSO

compile_seqmap_identify_shift_registers(3)
compile_variables(3)
set_case_analysis(2)
set_dont_touch_network(2)
set_ideal_network(2)

compile_seqmap_propagate_constants

Controls whether the **compile** command tries to identify and remove constant registers and propagate the constant value throughout the design.

TYPE

Boolean

DEFAULT

true

GROUP

compile_variables

DESCRIPTION

When true (the default), **compile** tries to identify and remove constant sequential elements in the design. In doing so, **compile** will improve the area of the design. When a constant register is removed, an OPT-1206 message is printed. A register is considered to be constant if it is forced into a state (0 or 1) and can never escape that state. A register is also considered to be constant if it cannot escape its reset state. The latter behavior is further controlled by the variable compile_seqmap_propagate_high_effort. The reset state of a register is determined by the presence of set and/or reset inputs on that registers. For example, if a register has a non-constant reset input and no set input it is assumed that the reset state of the register is logic zero.

To determine the current value of this variable use **printvar compile_seqmap_propagate_constants**. For a list of all **compile** variables and their current values, use the **print_variable_group compile** command.

SEE ALSO

compile_variables(3)
compile_seqmap_propagate_high_effort(3)

compile_seqmap_propagate_high_effort

Controls whether the **compile** command considers registers that cannot escape their reset state to be constant.

TYPE

Boolean

DEFAULT

true

GROUP

compile_variables

DESCRIPTION

When true and when compile_seqmap_propagate_constants is true, **compile** tries to identify registers that cannot escape their reset state. Such registers are considered to be constant and are removed from the design, thus improving the area of the design.

The reset state of a register is determined by the presence of set and/or reset inputs on that register. For example, if a register has a non-constant reset input and no set input it is assumed that the reset state of the register is logic zero. Furthermore, if once this register is in the logic zero state, it cannot enter the logic one state, it is considered to be trapped in the logic zero state and will be removed by **compile** as a constant register.

compile prints an OPT-1206 message whenever a constant register is removed from the design.

To determine the current value of this variable use **printvar compile_seqmap_propagate_high_effort**. For a list of all **compile** variables and their current values, use the **print_variable_group compile** command.

SEE ALSO

compile_variables(3)
compile_seqmap_propagate_constants(3)

compile_slack_driven_buffering

Controls whether rule based optimization will run additional steps of slack-driven buffering in ultra mode.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

Set this variable to *true* to enable the slack-driven buffering during **compile_ultra**. Set **compile_force_slack_driven_buffering** if you want to force slack-driven buffering while running the **compile** command. The variable **compile_force_slack_driven_buffering** takes priority over **compile_slack_driven_buffering**. Enabling slack-driven buffering usually results in slightly better worst negative slack in post-placement netlists, at the expense of runtimes.

SEE ALSO

compile_variables(3)
compile(2)
compile_ultra(2)

compile_top_acs_partition

Controls whether the **compile -top** command only fixes all violations that cross top-level partitions.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

This variable is for use only with the **-top** option of **compile**.

When true, **compile -top** fixes all design rule violations and all timing violations that cross the top-level partitions. When false (the default), **compile -top** fixes all design rule violations, but only those timing violations that cross top-level hierarchical boundaries.

To determine the current value of this variable, use **printvar compile_top_acs_partition**.

For a list of all **compile** variables and their current values, use the **print_variable_group compile** command.

SEE ALSO

compile(2)
set_compile_partitions(2)
compile_variables(3)

compile_top_all_paths

Controls whether the **compile -top** command fixes all violations in the design, or only those that cross top-level hierarchical boundaries.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

This variable is for use only with the **-top** option of **compile**.

When true, **compile -top** fixes all design rule violations and all timing violations present in the design. When false (the default), **compile -top** fixes all design rule violations, but only those timing violations that cross top-level hierarchical boundaries.

To determine the current value of this variable, use **printvar compile_top_all_paths**.

For a list of all **compile** variables and their current values, use the **print_variable_group compile** command.

SEE ALSO

compile(2)
compile_variables(3)

compile_ultra_ungroup_dw

Determines whether to unconditionally ungroup DesignWare cells in compile_ultra flow.

TYPE

Boolean

DEFAULT

true

GROUP

compile_variables

DESCRIPTION

In compile_ultra flow, by default, all designware hierarchies are unconditionally ungrouped in the second pass of compile.

The **compile_ultra_ungroup_dw** variable Determines whether the DesignWare cells will be auto-ungrouped in the compile_ultra flow. If set to false, compile_ultra will not unconditionally ungroup the DesignWare cells.

To determine the current value of this variable, type **printvar compile_ultra_ungroup_dw**. For a list of all **compile** variables and their current values, type **print_variable_group compile**.

SEE ALSO

compile_ultra(2)

compile_ultra_ungroup_small_hierarchies

Determines whether to automatically ungroup small hierarchies in the compile_ultra flow.

TYPE

Boolean

DEFAULT

true

GROUP

compile_variables

DESCRIPTION

In the compile_ultra flow in XG mode, small user design hierarchies are automatically ungrouped at the beginning of the compile flow. To turn off this behavior, set the above variable to false, or run **compile_ultra** with the **-no_autoungroup** option.

Use the **printvar compile_ultra_ungroup_small_hierarchies** command to determine the current value of this variable.

SEE ALSO

compile_ultra(2)

compile_update_annotated_delays_during_inplace_opt

Controls whether **compile -in_place** can update annotated delay values in the neighborhood of swapped cells. It has no effect for **reoptimize_design** and **physopt**, which always update annotated delay values.

TYPE

Boolean

DEFAULT

true

GROUP

compile_variables
links_to_layout_variables

DESCRIPTION

When true (the default value), **compile -in_place** is allowed to modify the values of annotated delays on nets connected to the swapped cells and to remove annotated delays on cells connected to the swapped cells.

When false, **compile -in_place** disallows annotated delays to be modified.

This variable is used to run **compile -in_place** with annotated cell and net delays and allows the optimization to update the annotated delays to reflect the timing changes due to swapping cells.

For example, when a cell is swapped for a cell whose input pins have higher pin capacitances, the transition delay on the cells in the fanin increases and invalidates the delays annotated on these cells. In the same manner, swapping a cell will modify the slope component of the delay of cells in the fanout of the swapped cell, thus invalidating the delay annotated on these cells.

With this variable set to true, the annotated delays on cells in the fanin and fanout of the swapped cells are removed so that new cell delays will be calculated. It is important to back-annotate net resistances and especially net capacitances to get accurate cell delays.

With this variable set to true, the annotated delays on nets connected to the swapped cells are incrementally modified to take into account the change in load due to the new pin capacitance of the new cells.

Set this variable to false to disallow modification of annotated net delays and removal of annotated cell delays.

To determine the current value of this variable, type **printvar compile_update_annotated_delays_during_inplace_opt**. For a list of all **compile** variables and their current values, type **print_variable_group compile**. For a list of all **links_to_layout** variables and their current values, type **print_variable_group links_to_layout**.

SEE ALSO

compile(2), compile_variables(3). links_to_layout_variables (3).

compile_use_fast_delay_mode

Selects the algorithm used for delay calculations when using the CMOS2 or nonlinear delay models.

TYPE

Boolean

DEFAULT

true

GROUP

compile_variables

DESCRIPTION

When true, turns on delay calculation techniques that improve **compile** run times when the CMOS2 or nonlinear delay models are being used. The default is true. The improvements have the greatest impact when high fanout nets are encountered, as is often the case during sequential mapping.

Unlike the delay calculation techniques used in **compile_use_low_timing_effort**, the corresponding techniques used in **compile_use_fast_delay_mode** do not affect timing accuracy. Fast delay mode may be used in conjunction with low timing effort, if desired.

To determine the current value of this variable, type **printvar compile_use_fast_delay_mode**. For a list of all **compile** variables and their current values, type **print_variable_group compile**.

SEE ALSO

compile(2)
compile_use_low_timing_effort(3)
compile_variables(3)

compile_variables

SYNTAX

Boolean **auto_wire_load_selection** = "true"
Boolean **compile_assume_fully_decoded_three_state_busses** = "false"
integer **compile_auto_ungroup_area_num_cells** = 30
Boolean **compile_auto_ungroup_count_leaf_cells** = "false"
Boolean **compile_auto_ungroup_override_wlm**= "false"
string **compile_automatic_clock_phase_inference** = "strict"
Boolean **compile_checkpoint_phases** = "false"
float **compile_cpu_limit** = "0.0"
Boolean **compile_create_wire_load_table** = "false"
Boolean **compile_delete_unloaded_sequential_cells** = "true"
Boolean **compile_disable_area_opt_during_inplace_opt** = "false"
Boolean **compile_disable_hierarchical_inverter_opt** = "false"
Boolean **compile_dont_touch_annotated_cell_during_inplace_opt** = "false"
Boolean **compile_fix_cell_degradation** = "false"
Boolean **compile_hold_reduce_cell_count** = "false"
Boolean **compile_ignore_area_during_inplace_opt** = "false"
Boolean **compile_ignore_footprint_during_inplace_opt** = "false"
Boolean **compile_implementation_selection** = "true"
string **compile_instance_name_prefix** = "U"
string **compile_instance_name_suffix** = ""
string **compile_log_format** = "%elap_time %area %wns %tns %drc %endpoint"
Boolean **compile_negative_logic_methodology** = "false"
Boolean **compile_no_new_cells_at_top_level** = "false"
Boolean **compile_ok_to_buffer_during_inplace_opt** = "false"
Boolean **compile_preserve_subdesign_interfaces** = "false"
Boolean **compile_seqmap_propagate_constants** = "false"
Boolean **compile_top_all_paths** = "false"
Boolean **compile_update_annotated_delays_during_inplace_opt** = "true"
Boolean **compile_use_fast_delay_mode** = "true"
Boolean **compile_use_low_timing_effort** = "false"
string **default_port_connection_class**   "universal"
Boolean **enable_recovery_removal_arcs** = "false"
string **port_complement_naming_style** = "%s_BAR"
string **reoptimize_design_changed_list_file_name** = ""
float **rtl_load_resistance_factor** = 0.0
Boolean **hlo_disable_datapath_optimization** = "false"
Boolean **compile_clock_gating_through_hierarchy** = "true"

DESCRIPTION

These variables directly affect the **compile** command.

For a list of these variables and their current values, type **print_variable_group compile**. To view this manual page online, type **help compile_variables**. To view an individual variable description, type **help *var***, where *var* is the variable name.

auto_wire_load_selection
    When *true* (the default value), turns on the automatic selection of the wire load model.

compile_assume_fully_decoded_three_state_busses

When *true*, **compile** and **translate** assume that three-state busses are fully decoded and therefore can be replaced by multiplexed busses when mapping to a library that contains no three-state cells. The default is *false*.

compile_auto_ungroup_area_num_cells

Provides control over the size of the hierarchy suitable for area-based automatic ungrouping during **compile -auto_ungroup area**. Its default value is 30.

compile_auto_ungroup_override_wlm

Specifies if a cell instance should be considered for auto_ungroup during compile if its wire load model differs from that of its parent. Its default value is false.

compile_automatic_clock_phase_inference

When set to *strict*, **compile** will attempt to determine the desired clock phase for each unmapped register, and will not allow opposite phase devices to be used in constructing registers. When set to *relaxed*, **compile** will allow the implementation of an opposite phase device for a register only if there is no other way to implement that register. When set to *none*, **compile** will ignore clock phase during sequential mapping. The default is *strict*.

compile_checkpoint_phases

When *true*, checkpoints automatically between each phase of **compile**. The default is *false*.

compile_cpu_limit

Specifies a time, in minutes, to be used as the limit for the amount of time to be spent in the phases after structuring and mapping. Optimization aborts when the limit is reached. The default value, "0.0", indicates that there is no limit.

compile_create_wire_load_table

Controls the type of wire load model generated by the **create_wire_load** command. When *true*, the wire load models generated are in the wire_load_table format. The default setting of this variable is *false* and the wire load models generated are in the wire_load format. It includes resistance, capacitance, area slope coefficients, and fanout_length (fanout, length, average_cap, std_dev, and points)

compile_delete_unloaded_sequential_cells

A design can contain sequential cells that drive no loads. During **compile**, the logic driven by a sequential cell might be optimized away, resulting in an inferred no-load or no path to any primary output. By default, **compile** deletes such sequential cells. To retain such cells, set variable **compile_delete_unloaded_sequential_cells** to *false*

compile_disable_area_opt_during_inplace_opt

When *true*, disables area optimization during inplace optimization. When *false* (the default value), area optimization is enabled; that is, **compile -in_place** attempts to save design area by downsizing cells that are not a part of the critical path. Note that this downsizing can sometimes help speed up the critical path. Thus, setting this option to *true*, while reducing the number of changes made to the design, may hinder this command from achieving the

best timing optimization results.
Note: This variable no longer works for **reoptimize_design**.

compile_disable_hierarchical_inverter_opt
When *true*, disables hierarchical inverter optimization. When *false* (the default setting), boundary optimization pushes inverters across hierarchy to improve the optimization cost of the design.

compile_dont_touch_annotated_cell_during_inplace_opt
When *true*, **reoptimize_design -in_place** and **compile -in_place** disallow swapping cells that have annotated delays. When *false* (the default value), **reoptimize_design -in_place** and **compile -in_place** allow annotated cells to be swapped for cells without annotated delay.

compile_fix_cell_degradation
When *true*, the algorithms for fixing cell_degradation violations in **compile** and **reoptimize_design** are activated. Different strategies, such as sizing and buffering, try to fix violations of the cell_degradation design rule.

compile_hold_reduce_cell_count
When *true* Design Compiler uses the minimum number of cells to fix the hold time (min path) violations, rather than choosing cells that minimize the total new area. This means that the area may be worsened (compared to the default flow) while the hold time violations and being fixed.

compile_ignore_area_during_inplace_opt
When *true*, **compile -in_place** is allowed to swap cells that have the same number of pins, pin names, logic functionality, and footprint, regardless of area. When *false* (the default value), cells are not swapped if they have different areas.
Note: This variable no longer works for **reoptimize_design**.

compile_ignore_footprint_during_inplace_opt
When *true*, **compile -in_place** is allowed to swap cells that have the same number of pins, pin names, logic functionality, and cell area, regardless of footprint. When *false* (the default value), cells are not swapped if they have different footprints.
Note: This variable no longer works for **reoptimize_design**.

compile_implementation_selection
When *true* (the default), **compile** re-evaluates the current implementation of a synthetic library module and replaces it if appropriate for optimizing the design.

compile_instance_name_prefix
Specifies the prefix used in generating cell instance names when **compile** is executed.

compile_instance_name_suffix
Specifies the suffix used for generating cell instance names when **compile** is executed.

compile_log_format
Controls the format of the columns to be printed during the mapping phases of **compile** and **reoptimize_design**.

compile_mux_optimization

When *true* (the default), **compile** takes advantage of dont-care inputs, constant inputs and shared intermediate terms while optimizing MUX_OP implementations. When *false* these optimizations are disabled.

compile_negative_logic_methodology

When *true*, **compile** and **translate** connect floating inputs to logic 1. When *false* (the default value), floating inputs are connected to logic 0.

compile_no_new_cells_at_top_level

When *true*, no new cells are added to the top-level design of the hierarchy during **compile**. New cells are added only to lower levels. The default is *false*.

compile_ok_to_buffer_during_inplace_opt

When *true*, **compile -in_place** is allowed to add buffers (or two inverters in a row) to the design to help meet timing constraints and fix design rule violations, including minimum path timing violations. The default is *false*. **Note:** This variable no longer works for **reoptimize_design**.

compile_preserve_subdesign_interfaces

When *true*, disables customization of logic external to a subdesign during **compile**, and preserves the subdesign interface. When *false* (the default), **compile** customizes the logic external to a subdesign based on the subdesign's internal logic.

compile_update_annotated_delays_during_inplace_opt

When *true* (the default value), **reoptimize_design -in_place** and **compile -in_place** are allowed to modify the values of annotated delays on nets connected to the swapped cells and to remove annotated delays on cells connected to the swapped cells. When *false*, **reoptimize_design -in_place** and **compile -in_place** disallow annotated delays to be modified.

compile_use_fast_delay_mode

When *true*, turns on delay calculation techniques that improve **compile** run times when the CMOS2 or nonlinear delay models are being used. The default is *true*. The improvements have the greatest impact when high fanout nets are encountered, as is often the case during sequential mapping.

compile_use_low_timing_effort

When *true*, **compile** uses a simplified delay model that may speed up optimization. The default is *false*.

enable_recovery_removal_arcs

Recovery or removal timing arcs impose constraints on asynchronous pins of sequential cells. When *true*, enables **compile**, **report_timing**, and **report_constraint** to accept recovery or removal arcs specified in the library.

hlo_disable_datapath_optimization

When *true*, disables the built-in datapath optimization feature in **compile**. When *false* (the default) the datapath optimization feature is enabled.

port_complement_naming_style

Defines the convention used by **compile** to rename ports complemented as a result of **set_boundary_optimization**.

reoptimize_design_changed_list_file_name
: The name of a file to which **reoptimize_design -in_place** and **reoptimize_design -post_layout_opto** commands should output a list of design modifications/ additions.

rtl_load_resistance_factor
: Specifies a factor to be used by the **set_rtl_load** command to calculate resistance values from capacitance values for RTL loads. The default is 0.0.

compile_top_all_paths
: If this variable is set to *true*, the **compile -top** command corrects all design rule violations and all timing violations present in the design.

compile_clock_gating_through_hierarchy
: Controls whether the **compile** or **compile_ultra** command with the **-gate_clock** option will perform clock gating through hierarchy

SEE ALSO

compile (2), **reoptimize_design** (2), **set_boundary_optimization** (2), **set_rtl_load** (2), **set_wire_load** (2), **translate** (2), **uniquify** (2).

complete_mixed_mode_extraction

Enables extraction of both routed and unrouted nets with a single command. This is known as mixed-mode extraction.

TYPE

Boolean

DEFAULT

true

GROUP

physopt

DESCRIPTION

By default, the extract_rc command skips unrouted or partially routed nets and extracts only completely routed nets. To estimate partially routed or unrouted nets, you must use extract_rc -estimate. Consequently, you must run both extract_rc and extract_rc -estimate to completely extract a partially routed design.

Mixed-mode extraction simplifies this process by enabling simultaneous handling of routed and unrouted nets. To enable this capability in IC Compiler, set the complete_mixed_mode_extraction variable to true. When this variable is true,

• The extract_rc command performs detailed extraction first, followed by estimation for broken or unrouted nets.

• When you run the write_parasitics command after doing mixed-mode extraction, it writes out mixed-mode parasitics.

To disable the mixed-mode extraction capability, set the value of this variable to false.

To see the current value of this variable, type

```
icc_shell-t> printvar complete_mixed_mode_extraction
```

context_check_status

Reports whether the context_check mode is enabled (read-only).

TYPE

Boolean

DEFAULT

false

GROUP

system_variables

DESCRIPTION

Reports whether the context_check mode is enabled (read-only). One of a pair of status variables, **syntax_check_status** and **context_check_status**, whose values are set by the syntax checker and not by the user. You examine these variables to determine the status of the syntax_check or context_check mode of the syntax checker. A value of *true* indicates that the mode is enabled. A value of *false* indicates that the mode is disabled. For example, a value of **context_check_status = true** indicates that the context_check mode is enabled. The two modes cannot be enabled simultaneously. These two status variables, **syntax_check_status** and **context_check_status**, allow you to determine whether one mode is enabled before you attempt to enable the other mode.

You enable or disable the syntax_check and context_check modes by executing the commands **syntax_check** or **context_check**. Alternatively, you can invoke **dc_shell**, **design_compiler**, and **dp_shell** in either the context_check or syntax_check mode by using the **-syntax_check** or **-context_check** options. For more information, refer to the man pages for these commands or to the *Design Compiler Reference Manual*.

To determine the value of this variable, type **printvar context_check_status** or **echo context_check_status**. For a list of **system** variables and their current values, type **print_variable_group system**.

SEE ALSO

dc_shell(1)
design_analyzer(1)
dp_shell(1)
context_check(2)
syntax_check(2)
syntax_check_status(3)
system_variables(3)

core_area_attributes

Contains attributes related to core area.

DESCRIPTION

Contains attributes related to core area.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified object. Specified with **list_attribute -class core_area -application**, the definition of attributes can be listed.

Core Area Attributes

area

Specifies area of a core area object.
The data type of **area** is double.
This attribute is read-only.

bbox

Specifies the bounding-box of a core area. The **bbox** is represented by a **rectangle**.
The format of a *rectangle* specification is {{llx lly} {urx ury}}, which specifies the lower-left and upper-right corners of the rectangle.
The data type of **bbox** is string.
This attribute is read-only.

boundary

Specifies point list of a core area's boundary.
The data type of **boundary** is string.
This attribute is read-only.

cell_id

Specifies Milkyway design ID in which a core area object is located.
The data type of **cell_id** is integer.
This attribute is read-only.

direction

Specifies the direction to orient the rows in a core area.
The valid values can be horizontal and vertical.
The data type of **direction** is string.
This attribute is read-only.

is_double_back

Specifies whether a core area contains pairs of rows with one row flipped in each pair.
The data type of **is_double_back** is boolean.
This attribute is read-only.

is_flip_first_row

Specifies whether to flip the first row at the bottom of a horizontal core area or at the left of a vertical core area.

The data type of **is_flip_first_row** is boolean.
This attribute is read-only.

is_start_first_row
Specifies whether the pairing of rows starts at the bottom of a horizontal core area or at the left of a vertical core area.
The data type of **is_start_first_row** is boolean.
This attribute is read-only.

name
Specifies name of a core area object.
The data type of **name** is string.
This attribute is read-only.

num_rows
Specifies number of rows inside a core area.
The data type of **num_rows** is integer.
This attribute is read-only.

object_class
Specifies object class name of a core area, which is **core_area**.
The data type of **object_class** is string.
This attribute is read-only.

object_id
Specifies object ID in Milkyway design file.
The data type of **object_id** is integer.
This attribute is read-only.

row_density
Specifies the ratio of total height (if it’s a horizontal core area) or width (otherwise) of rows to the height or width of a core area.
The data type of **row_density** is string.
This attribute is read-only.

tile_height
Specifies height of tile cell used in a core area.
The data type of **tile_height** is double.
This attribute is read-only.

tile_width
Specifies width of tile cell used in a core area.
The data type of **tile_width** is double.
This attribute is read-only.

SEE ALSO

get_attribute(2),
list_attribute(2),
report_attribute(2),
set_attribute(2).

create_clock_no_input_delay

Affects delay propagation characteristics of clock sources created by using the **create_clock** command.

Note: This variable will become obsolete. Please adjust your scripts accordingly.

TYPE

Boolean

DEFAULT

false

GROUP

timing_variables

DESCRIPTION

This variable affects delay propagation characteristics of clock sources created by using the **create_clock** command. When the **create_clock_no_input_delay** variable is false (the default value), clock sources used in the data path are established as timing start points. The clock sources in the design propagate rising delays on every rising clock edge and propagate falling delays on every falling clock edge. You can disable this behavior by setting the **create_clock_no_input_delay** variable to true.

SEE ALSO

create_clock(2)

current_design

Specifies the design being worked on. This variable is used by most of the Synopsys commands.

TYPE

string

DEFAULT

""

GROUP

system_variables

DESCRIPTION

Specifies the design being worked on. This variable is used by most of the Synopsys commands.

Until a design is read into the system as the current design, the value of **current_design** is "<<undefined>>". When one or more designs are read into the system, any of them can be made current by assigning the design name to this variable.

For example, if the design "real" is in dc_shell, to make it current type:

```
current_design = real
```

To determine the current value of this variable use **printvar current_design**. For a list of all **system** variables and their current values, use the **print_variable_group system** command.

db_load_ccs_data

The variable is obsolete and is not needed any more. The tool takes care of loading the CCS timing information automatically.

TYPE

Boolean

DEFAULT

false

GROUP

none

DESCRIPTION

SEE ALSO

read_db(2)
link(2)

dc_shell_mode

Reports the mode of the current dc_shell session.

TYPE

string

DEFAULT

tcl

DESCRIPTION

This variable is set to the mode of the application currently running. The value of this variable can be either of *default* or *tcl*. A value of *default* means that the application is running in dcsh default mode. A value of *tcl* means that the application is running in Tcl mode. The variable is read-only.

To determine the current value of this variable, use **printvar dc_shell_mode** in Tcl mode, or use **printvar dc_shell_mode** in dcsh.

dct_placement_ignore_scan

Sets the flag for the placer to ignore scan connections in Design Compiler topograpical.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

Setting the **dct_placement_ignore_scan** variable to true flags the placer to ignore scan connections, especially for a scan-stitched design. This is similar to **create_placement -ignore_scan** in IC Compiler.

SEE ALSO

compile_ultra(2)

dct_prioritize_area_correlation

Determines if optimization should prioritize area correlation between DC-T and ICC.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

The setting of **dct_prioritize_area_correlation** variable determines how DC-T compilation prioritizes area correlation between DC-T and ICC. When this variable is set to false, which is its default value, DC-T prioritizes timing correlation over area correlation between DC-T and ICC. When this variable is set to true, DC-T prioritizes area correlation over timing correlation between DC-T and ICC. In this mode, the compiled netlist will have better area correlation. It is likely that post DC-T and ICC area will be larger. The post DC-T timing will be worse but the post ICC timing should be better.

To determine the current value of this variable, type **dct_prioritize_area_correlation**. For a list of all **compile** variables and their current values, type **print_variable_group compile**.

SEE ALSO

compile_ultra(2)

ddc_allow_unknown_packed_commands

Causes the read_file command to attempt to read DDC files which contain packed commands which are unknown to the current version of **dc_shell**.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When this variable is *true* (the default), the read_file command will attempt to read a DDC file which contains embedded commands (constraints) which are unknown to the current version of **dc_shell**. This may allow the tool to read a DDC file which was written by a newer version of **dc_shell** which, for some reason, is not currently available.

If this feature is enabled then any unrecognized commands will be discarded, possibly resulting in the loss of important data. It is strongly recommended that DDC files be read by the same (or later) version of the tool as was used to write them out.

If the variable is set to *false* then any attempt to read a file with unknown embedded commands will result in a read failure and DDC-6 error message. You may wish to set this variable to *false* as a check against possible loss of constraint data.

It may not be possible to read certain DDC files written by newer versions of the tool, even with this variable set to *true*.

This variable is only evaluated during the read_file command. Unknown commands read in from DDC files while this variable is *true* will continue to be skipped even if the variable is subsequently set to *false*.

SEE ALSO

read_file(2)

default_input_delay

Specifies the global default input delay value to be used for environment propagation.

TYPE

float

DEFAULT

30

GROUP

acs_variables

DESCRIPTION

Specifies the global default input delay value to be used for environment propagation. This variable is used by the **derive_constraints** command.

To determine the current value of this variable, type **printvar default_input_delay**.

SEE ALSO

derive_constraints(2)

default_name_rules

Contains the name of a name rule to be used as a default by the **change_names** command, if the command's **-rules** option does not specify a *name_rules* value.

TYPE

string

DEFAULT

""

GROUP

system_variables

DESCRIPTION

Contains the name of a name rule to be used as a default by the **change_names** command, if the command's **-rules** option does not specify a *name_rules* value. The **change_names** command changes the names of ports, cells, and nets to conform to the rules specified by **default_name_rules**. The name_rule you assign to **default_name_rules** must already have been defined using **define_name_rules**. For information on the format and creation of name_rules, refer to the **define_name_rules** manual page.

To determine the current value of this variable, type **printvar default_name_rules**. For a list of all **system** variables and their current values, type **print_variable_group system**.

SEE ALSO

change_names(2)
define_name_rules(2)
system_variables(3)

default_output_delay

Specifies the global default output delay value to be used for environment propagation.

TYPE

float

DEFAULT

30

GROUP

acs_variables

DESCRIPTION

Specifies the global default output delay value to be used for environment propagation. This variable is used by the **derive_constraints** command.

To determine the current value of this variable, type **printvar default_output_delay**.

SEE ALSO

derive_constraints(2)

default_port_connection_class

Contains the value of the connection class to be assigned to ports that do not have a connection class assigned to them.

TYPE

string

DEFAULT

universal

GROUP

compile_variables

DESCRIPTION

Contains the value of the connection class to be assigned to ports that do not have a connection class assigned to them. The default value for **default_port_connection_class** is **universal**.

To determine the current value of this variable, type **printvar default_port_connection_class**. For a list of all **compile** variables and their current values, type **print_variable_group compile**.

SEE ALSO

set_connection_class(2)

default_schematic_options

Specifies options to use when schematics are generated.

TYPE

string

DEFAULT

-size infinite

GROUP

schematic_variables
view_variables

DESCRIPTION

Specifies options to use when schematics are generated. When set to "-size infinite" (default), the schematic for a design is displayed on a single page. (Used by the Design Analyzer.)

To determine the current value of this variable use **printvar default_schematic_options**. For a list of all **schematic** or **view** variables and their current values, use **print_variable_group schematic** or **print_variable_group view**.

design_attributes

DESCRIPTION

Contains attributes that can be placed on a design.

There are a number of commands used to set attributes. Most attributes, however, can be set with the **set_attribute** command. If the attribute definition specifies a **set** command, use it to set the attribute. Otherwise, use **set_attribute**. If an attribute is "read only," it cannot be set by the user.

To determine the value of an attribute, use the **get_attribute** command. To remove attributes, use the **remove_attribute** command.

For a more detailed explanation of an attribute, refer to the manual pages of the appropriate **set** command. For information on all attributes, refer to the **attributes** manual page.

Design Attributes

async_set_reset_q
: Establishes the value (0 or 1) that should be assigned to the q output of an inferred register if set and reset are both active at the same time. To be used with **async_set_reset_qn**. Use these attributes only if you have used the **one_hot** or **one_cold** attributes/directives in your HDL description *and* your technology library is written using pre-V3.0a syntax; *or* if your technology library does not use a consistent convention for q and qn when set and reset are both active. If a V3.0a or later syntax technology library is used, then, by default, if set and reset are both active at the same time Design Compiler will use the convention of the selected technology library (**target_library**). Set with **set_attribute**.
Note: If you are unsure whether your technology library uses V3.0a syntax, ask your ASIC vendor.

async_set_reset_qn
: Establishes the value (0 or 1) that should be assigned to the qn output of an inferred register if set and reset are both active at the same time. To be used with **async_set_reset_q**. Use these attributes only if you have used the **one_hot** or **one_cold** attributes/directives in your HDL description *and* your technology library is written using pre-V3.0a syntax; *or* if your technology library does not use a consistent convention for q and qn when set and reset are both active. If a V3.0a or later syntax technology library is used, then, by default, if set and reset are both active at the same time Design Compiler will use the convention of the selected technology library (**target_library**). Set with **set_attribute**.
Note: If you are unsure whether your technology library uses V3.0a syntax, ask your ASIC vendor.

balance_registers
: Determines whether the registers in a design are retimed during **compile**. When *true* (the default value), **compile** invokes the **balance_registers** command, which moves registers to minimize the maximum register-to-register delay. Set this attribute to *false*, or remove it, to disable this behavior.

Set with **set_balance_registers**. **Note:** You cannot verify designs using **compile -verify** while the **balance_registers** attribute is set to *true*. In addition, if your design contains generic logic, you should ensure that all components are mapped to cells from the library before setting the **balance_registers** attribute.

boundary_optimization

Enables **compile** to optimize across hierarchical boundaries. Hierarchy is ignored during optimization for designs with this attribute set to *true*. Set with **set_boundary_optimization**.

default_flip_flop_type

Specifies the default flip-flop type for the current design. During the mapping process, **compile** tries to convert all unmapped flip-flops to this type. If **compile** is unable to use this flip-flop, it maps these cells into the smallest flip-flop possible. Set with **set_register_type -flip_flop** *flip_flop_name*.

default_flip_flop_type_exact

Specifies the exact default flip-flop type for the current design. During the mapping process, **compile** converts unmapped flip-flops to the exact flip-flop type specified here. Set with **set_register_type -exact -flip_flop** *flip_flop_name*.

default_latch_type_exact

Specifies the exact default latch type for the current design. During the mapping process, **compile** converts unmapped latches to the exact latch type specified here. Set with **set_register_type -exact -latch** *latch_name*.

design_type

Indicates the bype of the **current_design**. Allowed values are **fsm** (finite state machine); **pla** (programmable logic array); **equation** (Boolean logic); or **netlist** (gates). This attribute is *read only* and cannot be set by the user.

dont_touch

Identifies designs that are to be excluded from optimization. Values are *true* (the default) or *false*. Designs with the **dont_touch** attribute set to *true* are not modified or replaced during **compile**. Setting **dont_touch** on a design has an effect only when the design is instantiated within another design as a level of hierarchy; setting **dont_touch** on the top-level design has no effect. Set with **set_dont_touch**.

flatten

When set to *true*, determines that a design is to be flattened during **compile**. By default, a design is not flattened. Set with **set_flatten**.

flatten_effort

Defines the level of CPU effort that **compile** uses to flatten a design. Allowed values are *low* (the default), *medium*, or *high*. Set with **set_flatten**.

flatten_minimize

Defines the minimization strategy used for logic equations. Allowed values are *single_output*, *multiple_output*, or *none*. Set with **set_flatten**.

flatten_phase

When *true*, allows logic flattening to invert the phase of outputs during **compile**. By default, logic flattening does not invert the phase of outputs. Used only if the **flatten** attribute is set. Set with **set_flatten**.

is_combinational

true if all cells of a design and all designs in its hierarchy are combinational. A cell is combinational if it is non-sequential or non-tristate and all of its outputs compute a combinational logic function. The **report_lib** command will report such a cell as not a black-box. This attribute is *read only* and cannot be set by the user.

is_dw_subblock

true if the object (a cell, a reference, or a design) is a DW subblock that was automatically elaborated. This attribute is *read only* and cannot be set by the user.
NOTE: DW subblocks that are manually elaborated will not have this attribute.

is_hierarchical

true if the design contains leaf cells or other levels of hierarchy. This attribute is read only and cannot be set by the user.

is_mapped

true if all the non-hierarchical cells of a design are mapped to cells in a technology library. This attribute is *read only* and cannot be set by the user.

is_sequential

true if any cells of a design or designs in its hierarchy are sequential. A cell is sequential if it is not combinational (if any of its outputs depend on previous inputs). This attribute is *read only* and cannot be set by the user.

is_synlib_module

true if the object (a cell, a reference, or a design) refers to an unmapped module reference or if the object is (or refers to) a design that was automatically elaborated from a synlib module or a synlib operator. This attribute is *read only* and cannot be set by the user.
NOTE: synlib modules that are manually elaborated will not have this attribute.

is_unmapped

true if any of the cells are not linked to a design or mapped to a technology library. This attribute is *read only* and cannot be set by the user.

local_link_library

A string that contains a list of design files and libraries to be added to the beginning of the **link_library** whenever a **link** operation is performed. Set with **set_local_link_library**.

map_only

When set to *true*, **compile** will attempt to map the object exactly in the target library, and will exclude the object from logic-level optimization (flattening and structuring). The default is *false*. Set with **set_map_only**.

max_capacitance

A floating point number that sets the maximum capacitance value for input, output, or bidirectional ports; or for designs. The units must be consistent with those of the technology library used during optimization. Set with **set_max_capacitance**.

max_dynamic_power

A floating point number that specifies the maximum target dynamic power for the **current_design**. The units must be consistent with those of the technology library. If this attribute is specified more than once for a design, the latest value is used. Set with **set_max_dynamic_power**.

max_leakage_power

A floating point number that specifies the maximum target leakage power for the **current_design**. The units must be consistent with those of the technology library. If this attribute is specified more than once for a design, the latest value is used. Set with **set_max_leakage_power**.

max_total_power

A floating point number that specifies the maximum target total power for the **current_design**. Total power is defined as the sum of dynamic and leakage power. If this attribute is specified more than once for a design, the latest value is used. Set with **set_max_total_power**.

min_porosity

Specifies the minimum porosity of the design. **compile -routability** and **reoptimize_design** ensure that the porosity of the design is greater than the specified value. Set with **set_min_porosity** .

minimize_tree_delay

When *true* (the default value), **compile** restructures expression trees in the **current_design** or in a list of specified designs, to minimize tree delay. Set this attribute to *false* for any designs that you do not wish to be restructured. Set with **set_minimize_tree_delay**.

model_map_effort

Specifies the relative amount of CPU time to be used by **compile** during modeling, typically for synthetic library implementations. Values are *low*, *medium*, and *high*, or *1*, *2*, and *3*. If **model_map_effort** is not set, the value of **synlib_model_map_effort** is used. Set with **set_model_map_effort**.

model_scale

A floating point number that sets the model scale factor for the **current_design**. Set with **set_model_scale**.

optimize_registers

When *true* (the default value), **compile** automatically invokes the Behavioral Compiler **optimize_registers** command to retime the design during optimization. Setting the attribute to *false* disables this behavior. You cannot execute **compile -verify** if **optimize_registers** is set to *true* on the design. Also, your design cannot contain generic logic at the instant **optimize_registers** is invoked during **compile**. Set with **set_optimize_registers**.

part

A string value that specifies the Xilinx part type for a design. For valid part types, refer to the Xilinx *XC4000 Databook*. Set with **set_attribute**.

resource_allocation

Indicates the type of resource allocation to be used by **compile** for the **current_design**. Allowed values are *none*, indicating no resource sharing; *area_only*, indicating resource sharing with tree balancing without considering timing constraints; *area_no_tree_balancing*, indicating resource sharing without tree balancing and without considering timing constraints; and *constraint_driven* (the default), indicating resource sharing so that timing constraints are met or not worsened. The value of this attribute overrides the value of the variable **hlo_resource_allocation** for the **current_design**. Set with **set_resource_allocation**.

resource_implementation

Indicates the type of resource implementation to be used by **compile** for the **current_design**. Allowed values are *area_only*, indicating resource implementation without considering timing constraints; *constraint_driven*, indicating resource implementation so that timing constraints are met or not worsened; and *use_fastest*, indicating resource implementation using the fastest implementation initially and using components with smaller area later in uncritical parts of the design. The value of this attribute overrides the value of the variable **hlo_resource_implementation** for the **current_design**. Set with **set_resource_implementation**.

structure

Determines if a design is to be structured during **compile**. If *true*, adds logic structure to a design by adding intermediate variables that are factored out of the design's equations. Set with **set_structure**.

structure_boolean

Enables the use of Boolean (non-algebraic) techniques during the structuring phase of optimization. This attribute is ignored if the **structure** attribute is *false*. Set with **set_structure**.

structure_timing

Enables timing constraints to be considered during the structuring phase of optimization. This attribute is ignored if the **structure** attribute is *false*. Set with **set_structure**.

ungroup

Removes a level of hierarchy from the current design by exploding the contents of the specified cell in the current design. Set with **set_ungroup**.

wired_logic_disable

When *true*, disables the creation of wired OR logic during **compile**. The default is *false*; if this attribute is not set, wired OR logic will be created if appropriate. Set with **set_wired_logic_disable**.

wire_load_model_mode

Determines which wire load model to use to compute wire capacitance, resistance, and area for nets in a hierarchical design that has different wire load models at different hierarchical levels. Allowed values are *top*, use the wire load model at the top hierarchical level; *enclosed*, use the wire load model on the smallest design that encloses a net completely; and *segmented*, break the net into segments, one within each hierarchical level.

In the *segmented* mode, each net segment is estimated using the wire load model on the design that encloses that segment. The *segmented* mode is not supported for wire load models on clusters. If a value is not specified for this attribute, **compile** searches for a default in the first library in the link path. If none is found, *top* is the default. Set with **set_wire_load**.

SEE ALSO

get_attribute(2)
remove_attribute(2)
set_attribute(2)
attributes(3)

designer

Specifies the name of the current user.

TYPE

string

DEFAULT

""

GROUP

system_variables

DESCRIPTION

Specifies the name of the current user. This name is displayed on the schematics.

To determine the current value of this variable use **printvar designer**. For a list of all **system** variables and their current values, use the **print_variable_group system** command.

die_area_attributes

Contains attributes related to die area.

DESCRIPTION

Contains attributes related to die area.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified object. Specified with **list_attribute -class die_area -application**, the definition of attributes can be listed.

Die Area Attributes

bbox

Specifies the bounding-box of a die area. The **bbox** is represented by a **rectangle**.
The format of a *rectangle* specification is {{llx lly} {urx ury}}, which specifies the lower-left and upper-right corners of the rectangle.
The data type of **bbox** is string.
This attribute is read-only.

boundary

Specifies point list of a die area's boundary.
The data type of **boundary** is string.
This attribute is read-only.

cell_id

Specifies Milkyway design ID in which a die area object is located.
The data type of **cell_id** is integer.
This attribute is read-only.

name

Specifies name of a die area object.
The data type of **name** is string.
This attribute is read-only.

object_class

Specifies object class name of a die area, which is **die_area**.
The data type of **object_class** is string.
This attribute is read-only.

object_id

Specifies object ID in Milkyway design file.
The data type of **object_id** is integer.
This attribute is read-only.

SEE ALSO

get_attribute(2),
list_attribute(2),

report_attribute(2),
set_attribute(2).

disable_auto_time_borrow

Determines whether the **report_timing** command and other commands will use automatic time borrowing.

TYPE

Boolean

DEFAULT

false

GROUP

timing_variables

DESCRIPTION

Determines whether the **report_timing** command and other commands will use automatic time borrowing. When *false* (the default), automatic time borrowing occurs. Automatic time borrowing balances the slack along back-to-back latch paths, to reduce the overall delay cost. Allocating slack throughout the latch stages can improve optimization results.

When *true*, no slack balancing occurs during time borrowing. This means the first paths borrow enough time to meet the constraint until the *max_time_borrow* is reached. Setting the variable to *true* produces time-borrow results consistent with PrimeTime.

SEE ALSO

report_timing(2)
timing_variables(3)

disable_case_analysis

When true, disables constant propagation from both logic constants and **set_case_analysis** command constants.

TYPE

Boolean

DEFAULT

false

GROUP

timing

DESCRIPTION

When true, disables constant propagation from both logic constants and **set_case_analysis** command constants. By default, the variable **disable_case_analysis** is *false*, so constant propagation is performed if there has been a **set_case_analysis** command or if the variable **case_analysis_with_logic_constants** has been specified.

To determine the current value of this variable, use **printvar disable_case_analysis**.

SEE ALSO

remove_case_analysis(2)
report_case_analysis(2)
set_case_analysis(2)
case_analysis_with_logic_constants(3)

disable_library_transition_degradation

Controls whether the transition degradation table is used to determine the net transition time.

TYPE

Boolean

DEFAULT

false

GROUP

timing_variables

DESCRIPTION

When false (the default), **report_timing** and other commands that use timing in dc_shell use the transition degradation table in the library to determine the net transition time. When true, the timing commands behave as though there were no transition degradation across the net.

SEE ALSO

report_timing(2)
timing_variables(3)

disable_mdb_stop_points

Disable ’stop at any level’ functionality in write_mdb command.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Command write_mdb by default traverses the whole hierarchy for the given design and write all cell information into the destination milkyway DB. To increase the capacity and to reduce the run time, you can also specify some modules as ’stop point’ by setting attribute "is_mdb_stop_point" to true. Then write_mdb will treat these modules as leaf modules and will not explore their sub-modules.

This function can be turned off by setting **disable_mdb_stop_points** to true. Command write_mdb will ignore all ’stop_point’ when this variable is set to true.

EXAMPLE

```
set disable_mdb_stop_points true
```

SEE ALSO

set_attribute(2)

do_operand_isolation

Enables or disables operand isolation as a dynamic power optimization technique for a design.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

This variable enables or disables operand isolation as a dynamic power optimization technique for a design. When *false* (the default value), this technique is disabled. Set the value to *true* to enable operand isolation.

Operand isolation can insert isolation cells for both operators and hierarchical combinational cells. There are two mechanisms for operand isolation: automatic selection and user-driven mode. This and other parameters can be specified with the **set_operand_isolation_style** command.

To determine the current value of this variable, type **printvar do_operand_isolation**. For a list of power variables and their current values, type **print_variable_group power**.

SEE ALSO

set_operand_isolation_style(2)
set_operand_isolation_cell(2)
set_operand_isolation_scope(2)
set_operand_isolation_slack(2)
compile(2)
report_operand_isolation(2)
remove_operand_isolation(2)

dont_touch_nets_with_size_only_cells

Specifies whether a net is marked dont touch if it is driven by at least one cell marked size_only and drives at least one cell marked by size_only.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

When true, this variable indicates that all the nets that are driven by at least one cell marked size_only and drive at least one cell marked size_only, will be treated as if marked dont_touch. Cells can be marked size_only by using the **set_size_only** command. If a net is marked dont touch, it can not be changed by dc_shell commands that optimize or manipulate the design.

Default value of this variable is false.

To determine the current value of this variable use **printvar dont_touch_nets_with_size_only_cells**. For a list of **compile** variables and their current values, use the **print_variable_group compile** command.

SEE ALSO

compile_variables(3)
set_dont_touch(1)
set_size_only(1)

dpcm_arc_sense_mapping

Controls whether Design Compiler maps half unate arcs to preset and clear arcs for sequential cells.

TYPE

Boolean

DEFAULT

true

GROUP

dpcm_variables

DESCRIPTION

When true (the default), Design Compiler maps half unate arcs to preset/clear arcs for sequential cells. When false, Design Compiler leaves the arcs from DPCM as they are.

To determine the current value of this variable, type **printvar dpcm_arc_sense_mapping**. For a list of all **dpcm** variables and their current values, type **print_variable_group dpcm**.

SEE ALSO

dpcm_variables(3)

dpcm_debuglevel

Determines the level of debugging for Design Compiler.

TYPE

string

DEFAULT

0

GROUP

dpcm_variables

DESCRIPTION

Determines the level of debugging for Design Compiler. Must be set with an integer greater than or equal to 0. When set to level 0 (the default) or higher, Design Compiler calls DPCM to set the debug level. At level 1 or higher, Design Compiler turns on interrupt handling, which determines whether the tool terminated abnormally within DPCM. For level 3 or higher, Design Compiler displays all EXTERNAL calls made by DPCM.

To determine the current value of this variable, type **printvar dpcm_debuglevel**. For a list of all **dpcm** variables and their current values, type **print_variable_group dpcm**.

SEE ALSO

dpcm_variables(3)

dpcm_functionscope

Controls how DPCM determines the FunctionalMode value when doing timing calculations.

TYPE

string

DEFAULT

global

GROUP

dpcm_variables

DESCRIPTION

Allowed values are *global* (the default) and *instance*. When set to *global*, the value is cached in DPCM, and DPCM does not request FunctionalMode for every delay/slew calculation call. When set to *instance*, for every delay or slew calculation call, DPCM makes a call to the external function CurrentFunctionalMode. The behavior usually depends on the way DPCM is implemented.

To determine the current value of this variable, type **printvar dpcm_functionscope**. For a list of all **dpcm** variables and their current values, type **print_variable_group dpcm**.

SEE ALSO

dpcm_temperaturescope(3)
dpcm_wireloadscope(3)
dpcm_voltagescope(3)
dpcm_level(3)
dpcm_variables(3)

dpcm_level

Controls the mode used by DPCM for timing calculations.

TYPE

string

DEFAULT

performance

GROUP

dpcm_variables

DESCRIPTION

Allowed values are *performance* (the default) and *accurate*. When set to *performance*, Design Compiler requests DPCM to do calculations in performance mode. When set to *accurate*, Design Compiler requests DPCM to do calculations in accurate mode. Usually, fewer calls are made in performance mode, and DPCM caches more values.

To determine the current value of this variable, type **printvar dpcm_level**. For a list of all **dpcm** variables and their current values, type **print_variable_group dpcm**.

SEE ALSO

dpcm_temperaturescope(3)
dpcm_wireloadscope(3)
dpcm_functionscope(3)
dpcm_voltagescope(3)
dpcm_variables(3)

dpcm_libraries

Specifies the libraries of the **link_path** that use DPCM delay calculation.

TYPE

list

DEFAULT

""

GROUP

dpcm_variables

DESCRIPTION

Specifies the libraries of the **link_path** that use the DPCM delay calculation. By default, delay calculation is performed with the Synopsys library delays. Set this variable only if a DPCM library is available for the specified library.

The default value of **dpcm_libraries** is "".

To determine the current value of this variable, type **printvar dpcm_libraries**. For a list of all **dpcm** variables and their current values, type **print_variable_group dpcm**.

SEE ALSO

dpcm_version(3)
dpcm_rulepath(3)
dpcm_rulespath(3)
dpcm_tablepath(3)
dpcm_variables(3)

dpcm_rulepath

Specifies a list of paths, similar to a search path, for locating the DPCM main rule.

TYPE

list

DEFAULT

""

GROUP

dpcm_variables

DESCRIPTION

A list of paths, similar to a search path, for locating the DPCM main rule. Design Compiler sets it to the DCMRULEPATH unix variable used by the DPCM delay calculation. Set this variable only if a DPCM library is available for the libraries used by the current design, and use the path specified by the DPCM provider.

The default value of **dpcm_rulepath** is "".

To determine the current value of this variable, type **printvar dpcm_rulepath**. For a list of all **dpcm** variables and their current values, use the **print_variable_group dpcm** command.

SEE ALSO

dpcm_libraries(3)
dpcm_version(3)
dpcm_rulespath(3)
dpcm_tablepath(3)
dpcm_variables(3)

dpcm_rulespath

Locates the DPCM subrules when provided a list of paths, the list being similar to a search path.

TYPE

list

DEFAULT

""

GROUP

dpcm_variables

DESCRIPTION

Locates the DPCM subrules when provided a list of paths, the list being similar to a search path. Design Compiler sets it to the DCMRULESPATH UNIX variable used by the DPCM delay calculation. Set this variable only if a DPCM library is available for the libraries the current design uses. Use the path specified by the DPCM provider.

The default value of **dpcm_rulespath** is {}.

To determine the current value of this variable, type **printvar dpcm_rulespath**. For a list of all **dpcm** variables and their current values, type **print_variable_group dpcm**.

SEE ALSO

dpcm_libraries(3)
dpcm_rulepath(3)
dpcm_tablepath(3)
dpcm_variables(3)
dpcm_version(3)

dpcm_slewlimit

Determines Design Compiler behavior when input pin slew exceeds library limits.

TYPE

string

DEFAULT

true

GROUP

dpcm_variables

DESCRIPTION

Determines behavior when input pin slew exceeds library limits. When true (the default), Design Compiler limits the slew to the maximum slew. When false, Design Compiler allows DPCM to handle these cases.

To determine the current value of this variable, type **printvar dpcm_slewlimit**. For a list of all **dpcm** variables and their current values, type **print_variable_group dpcm**.

SEE ALSO

dpcm_variables(3)

dpcm_tablepath

Specifies a list of paths, similar to a search path, for locating the DPCM tables.

TYPE

list

DEFAULT

""

GROUP

dpcm_variables

DESCRIPTION

A list of paths, similar to a search path, for locating the DPCM tables. Design Compiler sets it to the DCMTABLEPATH unix variable used by the DPCM delay calculation. Set this variable only if a DPCM library is available for the libraries used by the current design, and use the path specified by the DPCM provider.

The default value of **dpcm_tablepath** is "".

To determine the current value of this variable, type **printvar dpcm_tablepath**. For a list of all **dpcm** variables and their current values, type **print_variable_group dpcm**.

SEE ALSO

dpcm_libraries(3)
dpcm_version(3)
dpcm_rulepath(3)
dpcm_rulespath(3)
dpcm_variables(3)

dpcm_temperaturescope

Controls whether DPCM requests Temperature for each delay calculation.

TYPE

string

DEFAULT

global

GROUP

dpcm_variables

DESCRIPTION

Allowed values are *global* (the default) and *instance*. When set to *global*, the value is cached in DPCM, and DPCM does not request Temperature for every delay/slew calculation call. When set to *instance*, for every delay/slew calculation call, DPCM makes a call to the external function CurrentTemperature. The behavior usually depends on the way DPCM is implemented.

To determine the current value of this variable, type **printvar dpcm_temperaturescope**. For a list of all **dpcm** variables and their current values, type **print_variable_group dpcm**.

SEE ALSO

dpcm_voltagescope(3)
dpcm_wireloadscope(3)
dpcm_functionscope(3)
dpcm_level(3)
dpcm_variables(3)

dpcm_variables

SYNTAX

```
string **dpcm_arc_sense_mapping = "TRUE"**
string **dpcm_debuglevel** = "0"
string **dpcm_functionscope** = "global"
string **dpcm_level** = "performance"
list **dpcm_libraries** = ""
list **dpcm_rulepath** = ""
list **dpcm_rulespath** = ""
string **dpcm_slewlimit** = "TRUE"
list **dpcm_tablepath** = ""
string **dpcm_temperaturescope** = "global"
string **dpcm_version** = "IEEE-P1481"
string **dpcm_voltagescope** = "global"
string **dpcm_wireloadscope** = "global"
```

DESCRIPTION

These variables affect the DPCM libraries and delay calculation in DPCM mode.

For a list of these variables and their current values, type **print_variable_group dpcm**. To view this manual page online, type **help dpcm_variables**. To view an individual variable description, type **help *var***, where *var* is the variable name.

dpcm_arc_sense_mapping

When *true* (the default), Design Compiler maps half unate arcs to preset/clear arcs for sequential cells. When *false*, Design Compiler leaves the arcs from DPCM as they are.

dpcm_debuglevel

This variable can be set to *0* or higher. Depending on the value, more debugging information is printed. The default is *0*.

dpcm_functionscope

Allowed values are *global* (the default) or *instance*. When set to *global*, the value is cached in DPCM, and DPCM does not request FunctionalMode for every delay/slew calculation call. When set to *instance*, for every delay or slew calculation call, DPCM makes a call to the external function CurrentFunctionalMode. The behavior usually depends on the way DPCM is implemented.

dpcm_level

Allowed values are *accurate* or *performance* (the default). When set to *performance*, DPCM uses peformance equations and caches various values (for example, wire load, voltage). When set to *accurate*, DPCM uses a different set of equations and does not cache values. The behaviour usually depends on the way DPCM is implemented.

dpcm_libraries

Specifies the libraries of the **link_path** that use the DPCM delay calculation. By default, delay calculation is performed with the Synopsys library delays.

The default is "". Set this variable only if a DPCM library is available for the specified library. There should be an equivalent .db file for each DPCM library.

dpcm_rulepath
A list of paths, similar to a search path, for locating the DPCM main rule. Design Compiler sets it to the DCMRULEPATH unix variable used by the DPCM delay calculation. Set this variable only if a DPCM library is available for the libraries used by the current design, and use the path specified by the DPCM provider. The default is "".

dpcm_rulespath
A list of paths, similar to a search path, for locating the DPCM subrules. Design Compiler sets it to the DCMRULEPATH unix variable used by the DPCM delay calculation. Set this variable only if a DPCM library is available for the libraries used by the current design, and use the path specified by the DPCM provider. The default is "".

dpcm_slewlimit
When *true* (the default), Design Compiler limits the slew to the maximum slew for input pins, when the slew is beyond the maximum limit. When *false*, Design Compiler does not limit the slew, and allows the DPCM to handle the situation.

dpcm_tablepath
A list of paths, similar to a search path, for locating the DPCM tables. Design Compiler sets it to the DCMTABLEPATH unix variable used by the DPCM delay calculation. Set this variable only if a DPCM library is available for the libraries used by the current design, and use the path specified by the DPCM provider. The default is "".

dpcm_temperaturescope
Allowed values are *global* (the default) or *instance*. When set to *global*, the value is cached in DPCM, and DPCM does not request Temperature for every delay/slew calculation call. When set to *instance*, for every delay or slew calculation call, DPCM makes a call to the external function CurrentTemperature. The behavior usually depends on the way DPCM is implemented.

dpcm_version
Specifies the version of the API used by the DPCM delay calculation. The supported DPCM API versions are *IEEE-P1481* (the default) and *IBM*.

dpcm_voltagescope
Allowed values are *global* (the default) or *instance*. When set to *global*, the value is cached in DPCM, and DPCM does not request RailVoltage values for every delay/slew calculation call. When set to *instance*, for every delay or slew calculation call, DPCM makes a call to the external function RailVoltage. The behavior usually depends on the way DPCM is implemented.

dpcm_wireloadscope
Allowed values are *global* (the default) or *instance*. When set to *global*, the value is cached in DPCM, and DPCM does not request Current WireLoadModel values for every delay/slew calculation call. When set to *instance*, for every delay or slew calculation call, DPCM makes a call to the external function CurrentWireLoadModel. The behavior usually depends on the way DPCM is

implemented.

SEE ALSO

dpcm_arc_sense_mapping (3), **dpcm_debug_level** (3), **dpcm_functionscope(3), dpcm_level** (3), **dpcm_libraries** (3), **dpcm_rulepath** (3), **dpcm_rulespath** (3), **dpcm_slewlimit** (3), **dpcm_tablepath(3), dpcm_temperaturescope** (3), **dpcm_version(3), dpcm_voltagescope** (3), **dpcm_wireloadscope** (3).

dpcm_version

Specifies the version of the API used by the DPCM delay calculation.

TYPE

string

DEFAULT

IEEE-P1481

GROUP

dpcm_variables

DESCRIPTION

Specifies the version of the API used by the DPCM delay calculation. The supported DPCM API versions are *IEEE-P1481* (the default) and *IBM*.

To determine the current value of this variable, type **printvar dpcm_version**. For a list of all **dpcm** variables and their current values, type **print_variable_group dpcm**.

SEE ALSO

dpcm_libraries(3)
dpcm_rulepath(3)
dpcm_rulespath(3)
dpcm_tablepath(3)
dpcm_variables(3)

dpcm_voltagescope

Controls whether DPCM requests RailVoltage values for every delay calculation.

TYPE

string

DEFAULT

global

GROUP

dpcm_variables

DESCRIPTION

Allowed values are *global* (the default) and *instance*. When set to *global*, the value is cached in DPCM, and DPCM does not request RailVoltage values for every delay/slew calculation call. When set to *instance*, for every delay or slew calculation call, DPCM makes a call to the external function RailVoltage. The behavior usually depends on the way DPCM is implemented.

To determine the current value of this variable, type **printvar dpcm_voltagescope**. For a list of all **dpcm** variables and their current values, type **print_variable_group dpcm**.

SEE ALSO

dpcm_temperaturescope(3)
dpcm_wireloadscope(3)
dpcm_functionscope(3)
dpcm_level(3)
dpcm_variables(3)

dpcm_wireloadscope

Controls whether DPCM requests WireLoadModel values for every delay calculation.

TYPE

string

DEFAULT

global

GROUP

dpcm_variables

DESCRIPTION

Allowed values are *global* (the default) and *instance*. When set to *global*, the value is cached in DPCM, and DPCM does not request Current WireLoadModel values for every delay/slew calculation call. When set to *instance*, for every delay or slew calculation call, DPCM makes a call to the external function CurrentWireLoadModel. The behavior usually depends on the way DPCM is implemented.

To determine the current value of this variable, type **printvar dpcm_wireloadscope**. For a list of all **dpcm** variables and their current values, type **print_variable_group dpcm**.

SEE ALSO

dpcm_temperaturescope(3)
dpcm_voltagescope(3)
dpcm_functionscope(3)
dpcm_level(3)
dpcm_variables(3)

duplicate_ports

Specifies whether ports are to be drawn on every sheet for which an input or output signal appears.

TYPE

Boolean

DEFAULT

false

GROUP

schematic_variables

DESCRIPTION

Partitioning option that specifies if ports are to be drawn on every sheet for which an input or output signal appears. When true, no off-sheet connectors are used for input and output signals, and signal ports are duplicated where indicated on each sheet.

To determine the current value of this variable use **printvar duplicate_ports**. For a list of all **schematic** variables and their current values, use the **print_variable_group schematic** command.

echo_include_commands

Controls whether the contents of a script file is printed as it executes.

TYPE

Boolean

DEFAULT

true

GROUP

system_variables

DESCRIPTION

When true, the **include** command prints the contents of files it executes. Otherwise, contents are not printed.

To determine the current value of this variable use **printvar echo_include_commands**. For a list of all **system** variables and their current values, use the **print_variable_group system** command.

edif_variables

List of global variables that affect the EDIF format **read**, **read_lib**, and **write** commands.

SYNTAX

```
string bus_dimension_separator_style =  "]["
string bus_extraction_style  =          "%s[%d:%d]"
Boolean bus_inference_descending_sort =  "true"
string bus_inference_style =      ""
string bus_naming_style =      "%s[%d]"
string bus_range_separator_style =   ":"
Boolean edifin_autoconnect_offpageconnectors = "false"
Boolean edifin_autoconnect_ports = "false"
string edifin_dc_script_flag = ""
Boolean edifin_delete_empty_cells = "true"
Boolean edifin_delete_ripper_cells = "true"
string edifin_ground_net_name = ""
string edifin_ground_net_property_name = ""
string edifin_ground_net_property_value = ""
string edifin_ground_port_name =     ""
string edifin_instance_property_name = ""
string edifin_lib_in_osc_symbol = ""
string edifin_lib_in_port_symbol = ""
string edifin_lib_inout_osc_symbol = ""
string edifin_lib_inout_port_symbol = ""
string edifin_lib_logic_0_symbol = ""
string edifin_lib_logic_1_symbol = ""
string edifin_lib_mentor_netcon_symbol = ""
string edifin_lib_out_osc_symbol = ""
string edifin_lib_out_port_symbol = ""
string edifin_lib_ripper_bits_property =
string edifin_lib_ripper_bus_end =
string edifin_lib_ripper_cell_name = ""
string edifin_lib_ripper_view_name = ""
string edifin_lib_route_grid = 1024
list edifin_lib_templates = {}
string edifin_portinstance_disabled_property_name = ""
string edifin_portinstance_disabled_property_value = ""
string edifin_portinstance_property_name = ""
string edifin_power_net_name = ""
string edifin_power_net_property_name = ""
string edifin_power_net_property_value = ""
string edifin_power_port_name =    ""
Boolean edifin_use_identifier_in_rename = "false"
string edifin_view_identifier_property_name = ""
string edifout_dc_script_flag = ""
string edifout_design_name = "Synopsys_edif"
string edifout_designs_library_name = "DESIGNS"
Boolean edifout_display_instance_names = "false"
Boolean edifout_display_net_names = "false"
Boolean edifout_external = "true"
string edifout_external_graphic_view_name =  "Graphic_representation"
```

string **edifout_external_netlist_view_name** =   "Netlist_representation"
string **edifout_external_schematic_view_name** = "Schematic_representation"
string **edifout_ground_name** = "logic_0"
string **edifout_ground_net_name** = ""
string **edifout_ground_net_property_name** = ""
string **edifout_ground_net_property_value** = ""
string **edifout_ground_pin_name** = "logic_0_pin"
string **edifout_ground_port_name** =   "GND"
string **edifout_instance_property_name** = ""
Boolean **edifout_instantiate_ports** = "false"
string **edifout_library_graphic_view_name** = "Graphic_representation"
string **edifout_library_netlist_view_name** = "Netlist_representation"
string **edifout_library_schematic_view_name** = "Schematic_representation"
Boolean **edifout_merge_libraries** = "false"
Boolean **edifout_multidimension_arrays** = "false"
Boolean **edifout_name_oscs_different_from_ports** = "false"
Boolean **edifout_name_rippers_same_as_wires** = "false"
Boolean **edifout_netlist_only** = "false"
Boolean **edifout_no_array** = "false"
Boolean **edifout_numerical_array_members** = "false"
string **edifout_pin_direction_in_value** = ""
string **edifout_pin_direction_inout_value** = ""
string **edifout_pin_direction_out_value** = ""
string **edifout_pin_direction_property_name** = ""
string **edifout_pin_name_property_name** = ""
string **edifout_portinstance_disabled_property_name** = ""
string **edifout_portinstance_disabled_property_value** = ""
string **edifout_portinstance_property_name** = ""
string **edifout_power_and_ground_representation** = "cell"
string **edifout_power_name** = "logic_1"
string **edifout_power_net_name** = ""
string **edifout_power_net_property_name** = ""
string **edifout_power_net_property_value** = ""
string **edifout_power_pin_name** = "logic_1_pin"
string **edifout_power_port_name** =   "VDD"
Boolean **edifout_skip_port_implementations** = false
string **edifout_target_system** = ""
Boolean **edifout_top_level_symbol** = "true"
string **edifout_translate_origin** = ""
string **edifout_unused_property_value** = ""
Boolean **edifout_write_attributes** = "false"
Boolean **edifout_write_constraints** = "false"
list **edifout_write_properties_list** = {}
Boolean **write_name_nets_same_as_ports** = "false"

DESCRIPTION

These variables directly affect the EDIF format **read**, **read_lib**, and **write** commands. The *EDIF Interface Application Note* contains more about these variables.

The *Mentor Interface Application Note* contains more information about Mentor interface variables.

For more about other format **read**, **read_lib**, and **write** commands variables, see

io_variables(3).

For more about Verilog and VHDL variables, see the *HDL Compiler for Verilog Reference*, *VHDL Compiler Reference*, respectively, and **hdl_variables(3).**

For a list of **edif** variables and their current values, type **print_variable_group edif**. To view this manual page online, type **help edif_variables**. To view an individual variable description, type **help *var***, where *var* is the name of the variable.

bus_dimension_separator_style

Affects the **read** command with the **edif**, **verilog**, or **vhdl** format options and the **write** command with the **edif** format option. With **read**, used with **bus_naming_style** to specify the naming style for a port, net, or cell instance member of a multidimensional EDIF array or a multidimensional Verilog or VHDL vector. With **write**, used with **bus_naming_style** and **bus_range_separator_style** to specify the naming style for a multidimensional port array or a net array in the EDIF file.

bus_extraction_style

Affects the **read** command with the **edif** format option. Used with **bus_naming_style** and **bus_dimension_separator_style** to specify the style used to name bus members created in **Design Compiler**.

bus_inference_descending_sort

Affects the **read** command with all format options except **db**, **verilog**, and **vhdl**; primarily used with the **lsi** option (LSI/NDL format). This variable specifies that the members of the port bus are to be sorted in descending order rather than in ascending order.

bus_inference_style

Affects the **read** command with all format options except **db**, **verilog**, and **vhdl**; primarily used with the **lsi** option (LSI/NDL format). This variable determines the style used to name inferred busses.

bus_naming_style

This variable affects the **read**, **write**, and **create_schematic** commands. For the **read** command with the **vhdl** or **verilog** format options, this variable controls naming of bit-blasted ports in **dc_shell** when they are created from multidimensional busses in the original source.

bus_range_separator_style

This variable affects the **write** and **create_schematic** commands. For the **write** command with the **edif** format option, this variable is used together with the **bus_naming_style** variable to generate the style for naming bus ranges. For the **create_schematic** command, this variable, together with the **bus_range_separator_style** variable, also controls naming of bussed nets, bussed ports and bussed rippers in the schematic. Works the same as for **write -f edif**.

edifin_autoconnect_offpageconnectors

When *true*, attaches an off-page connector to the port with the same name even if there is no explicit connection statement. If this variable is *false* (the default value), a file cannot be read at all if it contains an off-page connector and a port with the same name. Some EDIF writers do not include explicit connections between off-page connectors and ports, but do give them the same names.

edifin_autoconnect_ports
: When *true*, automatically connects a port to a net if they have the same name, even if there is no explicit connection statement. The default is *false*. Usually, this connection should not be made without an explicit connection statement. Some EDIF writers do not include explicit connections between ports and nets, but do give them the same names.

edifin_dc_script_flag
: When set to a non-empty string, sends commands, embedded as comments in the EDIF file, to **Design Compiler**. To indicate that a comment contains commands, the first argument in the "comment" construct must be the value of this variable. When this variable is not set, EDIF comments are ignored during **read**.

edifin_delete_empty_cells
: When *true* (the default value), deletes all designs that do not have cells or nets.

edifin_delete_ripper_cells
: When *true* (the default value), deletes all designs specified as cellType "RIPPER".

edifin_ground_net_name
: Specifies EDIF ground nets. If you set this variable to the name of an EDIF ground net, the net is recognized by Synopsys software as a ground net.

edifin_ground_net_property_name
: Used to specify EDIF ground nets. If you set this variable and **edifin_ground_net_property_value** equal to the property name and property value of an EDIF ground net, the net is recognized by Synopsys software as a ground net.

edifin_ground_net_property_value
: Specifies EDIF ground nets. If you set **edifin_ground_net_property_name** and this variable equal to the property name and property value of an EDIF ground net, the net is recognized by Synopsys software as a ground net.

edifin_ground_port_name
: Specifies EDIF ground ports. If you set this variable to the name of an EDIF ground port, the port is recognized by Synopsys software as a ground port.

edifin_instance_property_name
: If a property on an instance construct has this name, and if the property value is a string, that value is assigned to the **cell_property** attribute of the cell instance.

edifin_lib_in_osc_symbol
: Affects EDIF **read_lib** . If a cell with this name is found in the EDIF symbol library, it is declared as the name of the input off-sheet connector symbol in the Synopsys intermediate symbol library source file and Synopsys symbol library file.

edifin_lib_in_port_symbol

Affects EDIF **read_lib** . If a cell with this name is found in the EDIF symbol library, it is declared as the name of the input port symbol in the Synopsys intermediate symbol library source file and Synopsys symbol library file.

edifin_lib_inout_osc_symbol

Affects EDIF **read_lib.** If a cell with this name is found in the EDIF symbol library, it is declared as the name of the input/output off-sheet connector symbol in the Synopsys intermediate symbol library source file and Synopsys symbol library file.

edifin_lib_inout_port_symbol

Affects EDIF **read_lib.** If a cell with this name is found in the EDIF symbol library, it is declared as the name of the input/output off-sheet connector symbol in the Synopsys intermediate symbol library source file and Synopsys symbol library file.

edifin_lib_logic_0_symbol

Affects EDIF **read_lib** . If a cell with this name is found in the EDIF symbol library, it is declared as the name of the ground symbol in the Synopsys intermediate symbol library source file and Synopsys symbol library file.

edifin_lib_logic_1_symbol

Affects EDIF **read_lib** . If a cell with this name is found in the EDIF symbol library, it is declared as the name of the power symbol in the Synopsys intermediate symbol library source file and Synopsys symbol library file.

edifin_lib_mentor_netcon_symbol

Affects EDIF **read_lib** . If a cell with this name is found in the EDIF symbol library, it is declared as the name of the Mentor **$netcon** symbol in the Synopsys intermediate symbol library source file and Synopsys symbol library file.

edifin_lib_out_osc_symbol

Affects EDIF **read_lib.** If a cell with this name is found in the EDIF symbol library, it is declared as the name of the output off-sheet connector symbol in the Synopsys intermediate symbol library source file and Synopsys symbol library file.

edifin_lib_out_port_symbol

Affects EDIF **read_lib** . If a cell with this name is found in the EDIF symbol library, it is declared as the name of the output port symbol in the Synopsys intermediate symbol library source file and Synopsys symbol library file.

edifin_lib_ripper_bits_property

Affects EDIF **read_lib**. If a cell with the name specified by the variable edifin_lib_ripper_cell_name of cellType "RIPPER" is found in the EDIF symbol library, the value of this variable is assigned to the **ripped_bits_property** attribute of this ripper symbol in the Synopsys intermediate symbol library source file and Synopsys symbol library file. In EDIF files written with this ripper symbol, the value of this attribute is the name of a property that is placed on each ripper instance; the number of the bit (or range of bits) is the property’s value.

edifin_lib_ripper_bus_end
Affects EDIF **read_lib**. If a cell with the name specified by the variable **edifin_lib_ripper_cell_name** of cellType "RIPPER" is found in the EDIF symbol library, the value of this variable is assigned to the **ripped_pin** attribute of this ripper symbol in the Synopsys intermediate symbol library source file and Synopsys symbol library file. In EDIF files written with instances of this ripper symbol, this attribute specifies the "bus end" pin of the ripper. If no pin exists on the ripper symbol with the name specified by this variable, results might be erroneous. This information is used by **create_schematic** to identify the bus and wire ends of the ripper symbol.

edifin_lib_ripper_cell_name
Affects EDIF **read_lib**. If a cell with this name of cellType "RIPPER" is found in the EDIF symbol library, the values of the variables edifin_lib_ripper_bits_property and edifin_lib_ripper_bus_end are assigned to the **ripped_bits_property** and **ripped_pin** attributes of this ripper symbol in the Synopsys intermediate symbol library source file and Synopsys symbol library file.

edifin_lib_ripper_view_name
Affects EDIF **read_lib**. If a cell with the name specified by the variable edifin_lib_ripper_cell_name of cellType "RIPPER" is found in the EDIF symbol library, only the view with this name is added as this ripper symbol in the Synopsys intermediate symbol library source file and Synopsys symbol library file. If the value of this variable is not specified, only the first view of the cell is added as this ripper symbol.

edifin_lib_route_grid
Affects EDIF **read_lib**. The value of this variable is declared as the size of the route grid of the library in the Synopsys intermediate symbol library source file and Synopsys symbol library file.

edifin_lib_templates
Affects EDIF **read_lib**. This variable is a list of sublists. Each sublist contains three elements: first, a string that defines a sheet size name; second, a string that defines the template orientation (either "landscape" or "portrait"); third, a string that defines the symbol name corresponding to the sheet size specified by the first element.

edifin_portinstance_disabled_property_name
If the property of a portInstance construct has the same name as this variable string and the property's value is the same as **edifin_portInstance_disabled_property_value**, the **disabled** attribute is placed on the pin of the cell instance.

edifin_portinstance_disabled_property_value
If the property value of a portInstance construct is the same as this variable string, and if the property has the same name as **edifin_portInstance_disabled_property_name,** the **disabled** attribute is placed on the pin of the cell instance.

edifin_portinstance_property_name
If the property of a portInstance construct has the same name as this variable string, and if the property's value is a string, that value is assigned to the **pin_properties** attribute of the pin of the cell instance.

edifin_power_net_name

Specifies EDIF power nets. If you set this variable to the name of an EDIF power net, the net is recognized by Synopsys software as a power net.

edifin_power_net_property_name

If the values of this variable and **edifin_power_net_property_value** are the same as the property name and property value of an EDIF net, the net is recognized as a power net.

edifin_power_net_property_value

If the values of **edifin_power_net_property_name** and this variable are the same as the property name and property value of an EDIF net, the net is recognized as a power net.

edifin_power_port_name

Specifies EDIF power ports. If you set this variable to the name of an EDIF power port, the port is recognized by Synopsys software as a power port.

edifin_use_identifier_in_rename

When *true*, gives all objects created the name specified by the "identifier" in the "name" construct (which might be made up of a "rename" construct). When *false* (the default value), gives all created objects the name specified by the "name" in the "rename" construct (if the "name" construct is made up of a "rename" construct). Also affects EDIF **read_lib**.

edifin_view_identifier_property_name

Allows multiple views (representations) of a cell to be created in the Synopsys database. This variable specifies an EDIF property whose value becomes the name of the cell object created. When set to an empty string, only the first cell view is recognized and subsequent views are ignored. (Also affects EDIF **read_lib**.)

edifout_dc_script_flag

When set to a non-empty string, embeds comments containing Design Compiler commands into the EDIF file. This variable is used with **edifout_write_attributes** and **edifout_write_constraints**, which select the attributes and constraints to write out. The value of this variable is placed as the first argument of each of the "comment" constructs.

edifout_design_name

Controls the identifier in the "design" construct written at the end of EDIF files. If this variable is not set, the design identifier in the "design" construct is "**Synopsys_edif**". When you set this variable, any non-alphanumeric character in the string is changed to an underscore. This construct appears only at the end of an EDIF file written out.

edifout_designs_library_name

Specifies the name given to the EDIF library construct that contains the cell constructs for the designs being written. When set to an empty string, the name "**DESIGNS**" is used.

edifout_display_instance_names

When *true*, displays cell instance names in the schematic. The default is *false*.

edifout_display_net_names

When *true*, displays the net name at each port, pin, and off-sheet connector location in the schematic; the default is *false*. Two "display" constructs are included in the EDIF file for each port, pin, and off-sheet connector on a net.

edifout_external

When *true*, only the interface for each referenced symbol is written in an "external" form. (Symbol definitions are not included.) When *false* (the default value), complete definitions are written for all symbols used in an EDIF schematic.

edifout_external_graphic_view_name

Specifies the name given to the EDIF views used in libraries contained in an "external" construct in schematic EDIF files. This variable affects only symbols with no pins such as port or off-sheet connector symbols. Set this variable if your target EDIF reader expects specific view names for cells of type GRAPHIC.

edifout_external_netlist_view_name

Specifies the name given to the EDIF views used in libraries contained in an "external" construct in netlist EDIF files. Set this variable if your target EDIF reader expects specific view names for cells of type NETLIST.

edifout_external_schematic_view_name

Specifies the name given to the EDIF views used in libraries contained in an "external" construct in schematic EDIF files. This variable affects schematics and symbols, except for symbols with no pins such as port or off-sheet connector symbols. Set this variable if your target EDIF reader expects specific view names for cells of type SCHEMATIC.

edifout_ground_name

When **edifout_power_and_ground_representation** is set to *cell* and **edifout_netlist_only** is *true*, this is the name given to the ground cell. This variable is used in conjunction with **edifout_ground_pin_name**, which names the ground cell's pin.

edifout_ground_net_name

When **edifout_power_and_ground_representation** is set to *net* and this variable and the variable **edifout_power_net_name** are non-empty strings, the ground nets are written as one net. This variable specifies the name of the ground net.

edifout_ground_net_property_name

When **edifout_power_and_ground_representation** is set to *net* and this variable and the variables **edifout_ground_net_property_value**, **edifout_power_net_property_name**, and **edifout_power_net_property_value** are non-empty strings, this variable specifies the name of the property used to identify the ground nets (or the one ground net created by the variable **edifout_ground_net_name**).

edifout_ground_net_property_value

When **edifout_power_and_ground_representation** is set to *net* and this variable and the variables **edifout_ground_net_property_name**, **edifout_power_net_property_name**, and **edifout_power_net_property_value**, are non-empty strings, this variable specifies the value of the property used to identify the ground nets (or the one ground net created by the variable **edifout_ground_net_name**).

edifout_ground_pin_name

When **edifout_power_and_ground_representation** is set to *cell* and **edifout_netlist_only** is *true*, this is the name given to the pin of the ground cell.

edifout_ground_port_name

When **edifout_power_and_ground_representation** is set to *port*, two extra port constructs are included in the interface of every cell: the power port and the ground port. This is the name given to the ground port.

edifout_instance_property_name

If a cell instance has the **cell_property** attribute, a property with the same name as this variable and value the same as the **cell_property attribute**, is included on the EDIF instance.

edifout_instantiate_ports

When *true*, "symbol" constructs for ports and off-sheet connectors are included in a "library" (or "external") construct in an EDIF schematic. In addition, instantiations referencing those symbols are included in the "portImplementation" constructs for ports and off-sheet connectors in the "contents" constructs of EDIF schematics. When *false* (the default value), schematic descriptions are included in the "portImplementation" constructs for ports and off-sheet connectors in the "contents" constructs of EDIF schematics. This variable does not affect the way "portImplementation" constructs are written in "symbol" constructs.

edifout_library_graphic_view_name

Specifies the name given to the EDIF views used in libraries contained in a "library" construct (not an "external" construct) in schematic EDIF files. This variable affects only symbols with no pins such as port or off-sheet connector symbols. Set this variable if your target EDIF reader expects specific view names for cells of type GRAPHIC.

edifout_library_netlist_view_name

Specifies the name given to the EDIF views used in libraries contained in a "library" construct (not an "external" construct) in netlist EDIF files. Set this variable if your target EDIF reader expects specific view names for cells of type NETLIST.

edifout_library_schematic_view_name

Specifies the name given to the EDIF views used in libraries contained in a "library" construct (not an "external" construct) in schematic EDIF files. This variable affects schematics and symbols, except for symbols with no pins such as port or off-sheet connector symbols. Set this variable if your target EDIF reader expects specific view names for cells of type SCHEMATIC.

edifout_merge_libraries

When *true*, writes all cells for an EDIF file in the same library. This causes problems if there is more than one cell with the same name, but this variable is included for EDIF readers that can handle only a single library per file.

edifout_multidimension_arrays

When *true*, a bus will be represented as a multidimensional array if the structure of the bus so permits. The default is *false*.

edifout_name_oscs_different_from_ports

When *true*, each off-sheet connector that is connected to a port is given a different name from the port in the descriptions of designs written in EDIF format. The default value is *false*.

edifout_name_rippers_same_as_wires

When *true*, a bus ripper cell instance (in the EDIF file) is named the same as the wires extracted ("ripped") from the bus net connected to the bus-end pin of the ripper instance. When *false* (the default value), bus ripper cell instances are named in the order they are created during schematic generation. The name for the ripper instance is "Ripper_*n*", where *n* is the incrementing integer value for each bus ripper cell instance created.

edifout_netlist_only

When *true*, writes an EDIF netlist only. When *false* (the default value), writes an EDIF schematic in addition to the netlist.

edifout_no_array

When this variable and **edifout_netlist** are *true*, no array constructs are written. In designs containing busses, individual bus members are written instead. The default value is *false*.

edifout_numerical_array_members

When **edifout_netlist_only** is *true* and this variable is *true*, in member constructs the integerValue that is the index into an ascending array is also 0 for the first member of the array and also increases by 1 for each succeeding member of the array; the integerValue that is the index into a descending array is 0 for the member with the lowest numerical value of the array and increases by 1 for each member of the array of successively increasing numerical value. When **edifout_netlist_only** is *false* or this variable is *false* (the default value), in member constructs the integerValue that is the index into an array is 0 for the first member of the array and increases by 1 for each succeeding member of the array, regardless of whether the bus is ascending or descending.

edifout_pin_direction_in_value

Includes an EDIF property with the same name as **edifout_pin_direction_property_name**, and value the same as this variable string, on input pins.

edifout_pin_direction_inout_value

Includes an EDIF property with the same name as **edifout_pin_direction_property_name**, and value the same as this variable, on input/output pins.

edifout_pin_direction_out_value

Includes an EDIF property with the same name as **edifout_pin_direction_property_name**, and value the same as this variable, on output pins.

edifout_pin_direction_property_name
Includes an EDIF property on all pins. The name of the property is the same as this variable string. To determine the value of the property, this variable is used in conjunction with **edifout_pin_direction_in_value**, **edifout_pin_direction_inout_value**, and **edifout_pin_direction_out_value**.

edifout_pin_name_property_name
Includes on pins an EDIF property with the same name as this variable string, and value the same as the pin name.

edifout_portinstance_disabled_property_name
If the pin of a cell instance has the **disabled** attribute, a property with the same name as this variable string and a value the same as **edifout_portInstance_disabled_property_value** is created in the portInstance construct.

edifout_portinstance_disabled_property_value
If the pin of a cell instance has the **disabled** attribute, a property with the same name as **edifout_portInstance_disabled_property_name** and value the same as this variable string is created in the portInstance construct.

edifout_portinstance_property_name
If the **pin_properties** attribute is on the pin of a cell instance, a property with the same name as this variable string and value the same as the **pin_properties** attribute is created in the "portInstance" construct.

edifout_power_and_ground_representation
Determines power and ground representations in EDIF files. This variable can have the value *port*, *cell* (the default), or *net*.

edifout_power_name
When **edifout_power_and_ground_representation** is set to *cell*, and **edifout_netlist_only** is *true*, this is the name given to the power cell. This variable is used in conjunction with **edifout_power_pin_name**, which names the power cell's pin.

edifout_power_net_name
When **edifout_power_and_ground_representation** is set to *net*, and if this variable and the variable **edifout_ground_net_name** are non-empty strings, the power nets are written as one net. This variable specifies the name of the power net.

edifout_power_net_property_name
When **edifout_power_and_ground_representation** is set to *net*, and if this variable and the variables **edifout_ground_net_property_name**, **edifout_ground_net_property_value**, and **edifout_power_net_property_value** are non-empty strings, this variable specifies the name of the property used to identify the power nets (or the one power net created by the variable **edifout_power_net_name**).

edifout_power_net_property_value
When **edifout_power_and_ground_representation** is set to *net*, and if this variable and the variables **edifout_ground_net_property_name**, **edifout_ground_net_property_value**, and **edifout_power_net_property_name** are non-empty strings, this variable specifies the value of the property used to

identify the power nets (or the one power net created by the variable **edifout_power_net_name**).

edifout_power_pin_name

When **edifout_power_and_ground_representation** is set to *cell*, and **edifout_netlist_only** is *true*, this is the name given to the pin of the power cell.

edifout_power_port_name

When **edifout_power_and_ground_representation** is set to *port*, two extra port constructs are included in the interface of every cell: the power port and the ground port. This is the name given to the power port.

edifout_skip_port_implementations

When *true*, does not write "portImplementation" constructs for ports in the "contents" constructs of EDIF schematics. The default is *false*. Note that "portImplementation" constructs are still written in "symbol" constructs, regardless of the value of this variable.

edifout_target_system

Causes the values of related **edifout** variables to be overridden so that object names can be transferred to the target system, and to perform vendor-specific operations beyond the scope of **edifout** variables. Allowed values are *mentor*, *valid*, or *cadence*. For more details about *mentor* or *valid*, refer to the appropriate EDIF Interface Application Note.

edifout_top_level_symbol

When *true* (the default value), writes the top-level symbol in EDIF schematic files. To disable the top-level symbol, set this variable to *false*.

edifout_translate_origin

Transforms the values of coordinates in schematic descriptions to reflect the specified origin. Allowed values are *center*, which places the center of each sheet at the origin; or *lower-left*, which places the lower left corner of each sheet at the origin. Notice that when the lower left corner is at the origin, all x- and y-coordinates are non-negative.

edifout_unused_property_value

Used when writing out EDIF netlists to be processed by a Motorola EDIF reader. EDIF netlists are required to have the **unused** attribute attached to unconnected port instances (pins). The property value defined by this variable is attached to the property name defined by the variable **edifout_portInstance_property_name**. When this property/value pair satisfies the Motorola reader guidelines, the **unused** attribute is attached to the appropriate port instances.

edifout_write_attributes

When *true*, embeds comments containing **Design Compiler** attribute definitions into the EDIF file. The default is *false*. Definitions are written out for all attributes in the design that are active during the current **Design Compiler** session. This variable is used with **edifout_dc_script_flag**.

edifout_write_constraints

When *true*, embeds comments containing Design Compiler constraint commands into the EDIF file. The default is *false*. All active constraints on the design during the current Design Compiler session are written out. This variable is used in conjunction with **edifout_dc_script_flag**.

edifout_write_properties_list

Specifies a list of library, cell, or port properties to write into the EDIF description. The value of this variable is a list of strings containing the property names.

write_name_nets_same_as_ports

When *true*, nets that are connected to ports are given the same names as those ports in the descriptions of designs written in the EDIF, LSI, or TDL format. The default is *false*.

SEE ALSO

read (2), **read_lib** (2), **write** (2); **io_variables** (3), **hdl_variables** (3), **vhdlio_variables** (3).

enable_cell_based_verilog_reader

This variable turns on the verilog2cel verilog reader.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

By specifying this variable, user can enable verilog2cel verilog reader, which is a CEL based verilog reader and will save the netlist directly into the Milkyway CEL, without creating DC data structures.

To determine the current value of this variable, use **printvar enable_cell_based_verilog_reader**. For a list of all **hdl** variables and their current values, use the **print_variable_group hdl** command.

SEE ALSO

hdl_variables(3)
read_verilog(2)
mw_current_design(3)

enable_instances_in_report_net

Enables **report_net** to report on instances in the current design.

TYPE

Boolean

DEFAULT

true

GROUP

none

DESCRIPTION

To enable **report_net** to report on instances in the current design, set "enable_instances_in_report_net" to true.

Note that without this variable set to true, the **report_net** only reports nets for current design. Also, nets in the current instance are reported if current_instance is set.

SEE ALSO

report_net(2)

enable_page_mode

Controls whether long reports are displayed one page at a time (similar to the UNIX **more** command).

TYPE

Boolean

DEFAULT

false

GROUP

system_variables

DESCRIPTION

When true, long reports are displayed one page at a time (similar to the UNIX **more** command). Commands affected by this variable include **list**, **help**, and the **report** commands.

To determine the current value of this variable use **list enable_page_mode**. For a list of all **system** variables and their current values, use the **print_variable_group system** command.

enable_recovery_removal_arcs

Controls whether Design Compiler accepts recovery and removal arcs that are specified in the technology library.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

Recovery or removal timing arcs impose constraints on asynchronous pins of sequential cells. Typically, recovery time specifies the time the inactive edge of the asynchronous signal has to arrive before the closing edge of the clock. Removal time specifies the length of time the active phase of the asynchronous signal has to be held after the closing edge of clock.

To enable **compile**, **report_timing**, and **report_constraint** to accept recovery or removal arcs specified in the library, set **enable_recovery_removal_arcs** to true.

Note that independent of the value of this variable, commands **write_timing** and **report_delay_calculation** always accepts and reports recovery or removal timing information.

SEE ALSO

compile(2)
compile_variables(3)

enable_slew_degradation

Determines whether the transition degradation is taken into account for nets with physical information.

TYPE

Boolean

DEFAULT

true

GROUP

timing_variables

DESCRIPTION

When *true* (the default setting), timing calculation will take into account transition degradation across a net. For a long net, this may mean that the transition at the net driver could be very different than the various load pins. Because all load pins on the same net will not necessarily have the same transition time optimization with physical information will fix transitions on a per pin basis. This may lead to increased runtime for these commands.

Setting this variable to *false* will cause the timing calculation to not calculate transition degradation across a net. However, some libraries do have a transition degradation table which show how to degrade the transition across a net. If slew degradation is not enabled, then the library transition degradation tables are used unless the variable **disable_library_transition_degradation** is set to *true*. If neither slew degradation nor library transition degradation is enabled, the transition at the net driver will be the same transition at the load pin. Because all load pins on the same net have the same transition time, optimization with physical information will attempt to fix transitions on a per net basis.

Regardless of the setting of this variable, slew degradation will not occur for multi-driven nets. For multi-driven nets, there can be cases of too much pessimism.

Regardless of the setting of this variable, slew degradation can only occur if there are either delay back-annotation for the net or physical locations for the pins of the net. Otherwise, the transition time calculation at the driver pin is too inaccurate to be degraded.

To determine the current value of this variable, type **printvar enable_slew_degradation**. For a list of all **timing** variables and their current values, type **print_variable_group timing**.

SEE ALSO

disable_library_transition_degradation(3)

enable_special_level_shifter_naming

Setting this variable to true would enable special naming for automatically inserted level shifters

TYPE

Boolean

DEFAULT

false

GROUP

mv

DESCRIPTION

When this variable is set to true, automatically inserted level shifters by insert_level_shifters, compile and other commands are named specially. The name follows the template <prefix> + <PD OR Design name> + "_LS" + #. <prefix> is a user specifed prefix using the variable level_shifter_naming_prefix. # is a number internally generated to keep this name unique. <PD OR Design Name> is the name of power domain where the level shifter is being added, or the design name if the power domain is not defined

SEE ALSO

level_shifter_naming_prefix(3)

estimate_io_latency

Uses estimated I/O latency in timing calculations for ports when true.

TYPE

Boolean

DEFAULT

false

GROUP

timing

DESCRIPTION

When true, uses estimated I/O latency in timing calculations for ports.

For each input port the I/O latency for max case is defined as the minimum clock network latency for the clocks in the fanout set of the port. The I/O latency for min case is the maximum clock latency in the fanout set.

For each output port the I/O latency for max case is defined as the maximum clock network latency for the clocks in the fanin set of the port. The I/O latency for min case is the minimum clock latency in the fanin set.

The I/O latency will not be added to external delay in timing calculations if the network_delay_included switch is used. The ideal clocks are not used for I/O latency calculation.

This variable should ideally be used after clock tree synthesis.

To determine the current value of this variable, use **printvar estimate_io_latency**.

SEE ALSO

set_clock_latency(2)
set_propagated_clock(2)
report_timing(2)

exit_delete_command_log_file

Controls whether the file specified by the variable **command_log_file** is deleted after **design_analyzer** or dc_shell exits normally.

TYPE

string

DEFAULT

false

GROUP

system_variables

DESCRIPTION

When true, causes the file specified by the variable **command_log_file** to be deleted after **design_analyzer** or dc_shell exits normally. The default value is false. Set **exit_delete_command_log_file** to false if you want the file to be retained.

To determine the current value of this variable, type **printvar exit_delete_command_log_file**. For a list of all **system** variables and their current values, type **print_variable_group system**.

SEE ALSO

exit_delete_filename_log_file

Controls whether the file specified by the variable **filename_log_file** is deleted after **design_analyzer** or dc_shell exits normally.

TYPE

string

DEFAULT

true

GROUP

system_variables

DESCRIPTION

When true (the default value), causes the file specified by the variable **filename_log_file** to be deleted after **design_analyzer** or dc_shell exits normally. Set **exit_delete_filename_log_file** to false if you want the file to be retained.

To determine the current value of this variable, type **printvar exit_delete_filename_log_file**. For a list of all **system** variables and their current values, type **print_variable_group system**.

SEE ALSO

filename_log_file

Specifies the name of the filename log file to be used in case a fatal error occurs during execution of **design_analyzer** or dc_shell.

TYPE

string

DEFAULT

filenames.log

GROUP

system_variables

DESCRIPTION

Specifies the name of the filename log file to be used in case a fatal error occurs during execution of **design_analyzer** or dc_shell. The file specified by **filename_log_file** will contain all the filenames read in by **design_analyzer** or dc_shell, including db, script, verilog, vhdl or include files for one invocation of the program (however, the support to log filenames read in by **read_sdc** is not in yet). If there is a fatal error, you can easily identify the data files needed to reproduce the fatal error. If this variable is not specified, the default filename **filenames.log** will be used. This file will be deleted if the program exits normally, unless **exit_delete_filename_log_file** is set to false.

The variable is valid both in eqn mode as well as in tcl mode.

If the application has been invoked using "-no_log" switch, then the process id of the application and the timestamp is appended to filename log file, something like - filenames_<process_id>_<timestamp>.log.

To determine the current value of this variable, type **printvar filename_log_file**. For a list of all **system** variables and their current values, type **print_variable_group system**.

SEE ALSO

find_allow_only_non_hier_ports

Controls find command to search for ports in sub-designs (described as hier_ports here).

TYPE

Boolean

DEFAULT

false

GROUP

none

DESCRIPTION

When set to true, directs the find command to search for top level ports only, ignoring ports in sub-designs.

By default, **find port** will also fetch ports in the sub-designs.

For example, if cell 'c' is an instance of design SUB1, then with this variable set to false, the command "find port c/A" will fetch the port A on design SUB1. With the variable set to true, "find port c/A" will result in a warning message as below,

```
prompt> printvar find_allow_only_non_hier_ports
find_allow_only_non_hier_ports = "false"
prompt> find port c/A
{"c/A"}
prompt> find_allow_only_non_hier_ports = true
"true"
prompt> printvar find_allow_only_non_hier_ports
find_allow_only_non_hier_ports = "true"
prompt> find port c/A
Warning: Can't find port 'c/A' in design 'SUB1'. (UID-95)
```

This variable is to facilitate the netlist editing commands connect_net and disconnect_net which allow net and pin instances in addition to handling nets and pins in the current_design.

SEE ALSO

connect_net(2)
disconnect_net(2)
remove_port(2)
get_ports(2)
find(2)

find_converts_name_lists

Controls whether the **find** command converts the *name_list* string to a list of strings before searching for design objects.

TYPE

Boolean

DEFAULT

false

GROUP

system_variables

DESCRIPTION

When true, directs the **find** command to convert the *name_list* string to a list of strings before searching for design objects. In addition, when this variable is true, all commands that use the implicit **find** will convert appropriate strings to lists of strings before searching for objects. For example, **current_instance** uses the implicit **find** and would convert the string *instance* to a list of strings before searching for objects.

find_converts_name_lists provides backward compatibility with the pre-modification **find** command. When the variable is false (the default value), **find** executes according to its post-modification behavior; that is, strings are not converted to lists of strings.

To determine the current value of this variable, type **printvar find_converts_name_lists**. For a list of all **system** variables and their current values, type **print_variable_group system**.

SEE ALSO

find(2)
current_instance(2)
system_variables(3)

find_ignore_case

Controls whether the **find** command is case-sensitive when matching object names.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Normally, the **find** command is case-sensitive when matching names of objects. That is, for either an explicit or implicit invocation of the **find** command, **find** only matches objects whose names have the same case as the specified string.

However, sometimes it may be more desirable to use case-insensitive matching. The **find_ignore_case** variable provides that capability.

When the **find_ignore_case** variable is true, the **find** command performs case-insensitive string comparisons. This variable affects both implicit and explicit invocations of the **find** command. The default value for this variable is false.

To determine the current value of this variable, type **list find_ignore_case**. If the variable has not been defined yet, the **list** command will fail indicating that the variable is undefined.

Use this variable only when necessary, as it may slightly increase the run-time of find operations.

SEE ALSO

find(2)

floorplan_data_attributes

Contains attributes related to floorplan data on different modules in separate views.

DESCRIPTION

Contains attributes related to floorplan data on different modules in separate views.

Floorplan data attributes are read-only. You can use **get_floorplan_data** to determine value of an attribute, and use **report_floorplan_data** to get a report of all the floorplan data on module objects in a specified view. If you want to know the list of valid floorplan data names, you can use **list_floorplan_data**.

Floorplan Data Attributes

area
: Specifies the area of a module. This attribute does not exist on a logical module.
The tool calculates the attribute based on cell boundary.

aspect_ratio
: Specifies the **height**:**width** ratio of a module. This attribute does not exist on a logical module.

hard_macro_area_percentage
: Specifies ratio of sum of **area** of hard macros in a module to top module's **physical_area**.

height
: Specifies the height of a module. This attribute only exists on soft macros or hard macros in **logical** view.
The tool calculates the attribute based on the **bbox** of a module.

macro_area_percentage
: Specifies the ratio of sum of **area** of macros in a module to top module's **physical_area**.
Note that the definition of **macro_area_percentage** in this document is different from that in **cell_attributes.3**. Compared to the definition in **cell_attributes.3**, the numerator of ratio remains while the denominator is **physical_area** of top module instead of **physical_area** of the specified module.

number_of_black_box
: Specifies the count of black boxes in a module. This attribute does not exist on plan_group.
Whether a cell is a black box can be determined by the attribute **is_black_box**.

number_of_hard_macro
: Specifies the count of hard macros in a module.
Whether a cell is a hard macro can be determined by the attribute **is_hard_macro**.

number_of_io_cell
Specifies the count of io cells in a module. This attribute does not exist on **plan_group**.
The tool counts cells in when its **mask_layout_type** is **io_pad**, **corner_pad**, **pad_filler**, or **flip_chip_pad**.

number_of_macro
Specifies the count of macros in a module. This attribute does not exist on **plan_group**.
The tool counts cells in when its **mask_layout_type** is **macro**.

number_of_standard_cell
Specifies the count of standard cells in a module.
The tool counts cells in when its **mask_layout_type** matches ***std***.

physical_area
Specifies the physical area of a module.
The physical area of a hard macro is equal to its **area**.
In **one_level** or **logical** view, the physical area of a soft macro is equal to its **area**. However, in **physical** view, physical area is the sum of **physical_area** of macros and **area** of standard cells inside the specified soft macro.
The physical area of a plan group is sum of **area** of macros and standard cells in it.
For a top module or a logical module, rules for calculating **physical_area** are applied as:
In a **one_level** view, only children in the current level are counted and children should be either macro or standard cell. The physical area of a module in **one_level** is sum of the **area** of the counted children.
In a **logical** view, the hierarchy tree in the module is traversed and the **area** of macros and standard cells in the tree will be added.
In the **physical** view, the hierarchy tree in the module is traversed and the **area** of standard cells and **physical_area** of macros in the tree will be added.

physical_area_percentage_in_top_design
Specifies the ratio of **physical_area** of a module to top module's **physical_area**.

utilization
Specifies the utilization of a module.
The tool calculates the attribute using **physical_area**:**area** ratio of a module.
This attribute does not exist on a logical module or a top module.

width
Specifies the width of a module. This attribute only exists on soft macros or hard macros in **logical** view.
The tool calculates the attribute based on the **bbox** of a module.

SEE ALSO

get_floorplan_data(2)
list_floorplan_data(2)
report_floorplan_data(2)

fsm_auto_inferring

Determines whether or not to automatically extract finite state machine during the **compile**.

TYPE

Boolean

DEFAULT

false

GROUP

fsm_variables

DESCRIPTION

This variable determines whether the compiler performs finite state machine (FSM) automatic extraction during **compile**. If a design has previously had state machine extraction performed on it, the compiler does not perform extraction again with the same encoding style.

EXAMPLES

In the Presto flow, the compiler detects the FSM attributes from the HDL code and extracts the FSM during **compile**, as shown here:

```
   prompt> fsm_auto_inferring=true
   prompt> read -f verilog example.v
   prompt> set_fsm_encoding_style one_hot
   prompt> compile
```

SEE ALSO

set_fsm_encoding(2)
set_fsm_state_vector(2)
fsm_enable_state_minimization(3)
fsm_export_formality_state_info(3)

fsm_enable_state_minimization

Determines whether or not the state minimization is performed for all finite state machines (FSMs) in the design.

TYPE

Boolean

DEFAULT

false

GROUP

fsm_variables

DESCRIPTION

When this option is set to *true*, the compiler performs the state minimization after the state extraction. However, if the state number is changed, Formality and the newly developed FSM verification feature of the **compile -verify** command and the **compare_design -fsm** variable cannot verify the design. To pass the verification, set this option to *false*.

SEE ALSO

fsm_auto_inferring(3)
fsm_export_formality_state_info(3)

fsm_export_formality_state_info

Determines whether or not state machine encoding information is exported into the files that will be used by Formality.

TYPE

Boolean

DEFAULT

false

GROUP

fsm_variables

DESCRIPTION

When this variable is set to *true*, the state encoding information before and after finite state machine extraction is exported into two files, <module_name>.ref and <module_name>.imp, which will be used for Formality verification. Default value for this variable is *false*.

SEE ALSO

fsm_auto_inferring(3)
fsm_enable_state_minimization(3)

fsm_variables

Variables that affect finite state machine (FSM) optimization.

SYNTAX

```
Boolean fsm_auto_inferring = true
Boolean fsm_enable_state_minimization = false
Boolean fsm_export_formality_state_info = false
```

DESCRIPTION

These variables directly affect aspects of finite state machine (FSM) optimization in **dc_shell**. Defaults are shown under "SYNTAX."

For a list of fsm variables, type **print_variable_group fsm**. To view this manual page online, type **help fsm_variables**. To view an individual variable description, type **help *var***, where *var* is the name of the variable.

fsm_auto_inferring

Determines whether the compiler performs finite state machine (FSM) automatic extraction during **compile**. If a design has previously had state machine extraction performed on it, the compiler does not perform extraction again. The introduction of this new feature coincides with the old FSM compiler becoming obsolete.

fsm_enable_state_minimization

When this option is set to *true*, the compiler performs state minimization after state extraction. However, if the state number is changed, Formality and the newly developed FSM verification feature of the **compile -verify** command and the **compare_design -fsm** variable cannot verify the design. To pass the verification, set this option to *false*.

fsm_export_formality_state_info

When this variable is set to *true*, the state encoding information before and after FSM extraction is exported into two files, <module_name>.ref and <module_name>.imp, which are used for Formality verification. The default value of this variable is *false*.

fsm_extract_moore_machine

If this variable is ftruefp, the Moore finite state machines will be extracted.

fsm_set_safe_mode

If this variable is ftruefp, the safe-mode finite state machines will be extracted.
When a finite state machine(FSM) get into a invalid state for any reason, especially when power up. FSM may not possible returnng a valid state without reset. The safe-mode FSM is for sure the FSM can return from any invalid state to one of the valid states without reset. The aonther good feature is even with the safe-mode, the synthesised netlist can still be verified with Formality.

SEE ALSO

compare_design (2), **compile** (2), **report_fsm** (2).

fuzzy_matching_enabled

This is a Boolean variable used to enable or disable fuzzy matching.

TYPE

Boolean

DEFAULT

false

GROUP

none

DESCRIPTION

When this variable is set to true, fuzzy matching will be enabled.

If this variable is set to true without prior calling of 'set_query_option', the default fuzzy rules are used.

When this variable is set to true, runtime for query could be slower. Fuzzy matching should be disabled once it is no longer needed, especially when the netlist undergoes lots of changes.

SEE ALSO

set_fuzzy_query_options(2)

gen_bussing_exact_implicit

Controls whether schematics generated using the **create_schematic -implicit** command should contain implicit bus names instead of bus rippers.

TYPE

Boolean

DEFAULT

false

GROUP

schematic_variables

DESCRIPTION

When true, specifies that schematics generated using **create_schematic -implicit** should contain no bus rippers. Instead, all bused connections should be shown with implicit bus names. This should be used with the **-implicit** command line option. By default, schematics generated using the **-implicit** option have rippers between any bus connections where the ripper connects to a pin in a column adjacent to the originating bus pin. Bused connections between cell pins more than one column away from each other are always shown disconnected with the **-implicit** option.

To determine the current value of this variable, type **printvar gen_bussing_exact_implicit**. For a list of all **schematic** variables and their current values, type **print_variable_group schematic**.

SEE ALSO

schematic_variables(3)

gen_cell_pin_name_separator

Specifies the character used to separate cell names and pin names in the bus names generated by the **create_schematic** command.

TYPE

string

DEFAULT

/

GROUP

schematic_variables

DESCRIPTION

If this variable is set, then its value is used to separate the cell and pin names in the bus names generated by **create_schematic**. By default, "/" is used to separate the cell and pin names, thus creating bus names like "U0/OUT[0:3]"

To determine the current value of this variable, type **printvar gen_cell_pin_name_separator**. For a list of all **schematic** variables and their current values, type **print_variable_group schematic**.

SEE ALSO

gen_create_netlist_busses

Controls whether **create_schematic** creates netlist buses whenever it creates buses on the schematic.

TYPE

Boolean

DEFAULT

true

GROUP

schematic_variables

DESCRIPTION

Controls whether **create_schematic** creates netlist buses whenever it creates buses on the schematic. When true (the default), the **create_schematic** command does so. Usually this happens when **create_schematic** creates buses to connect to bused pins on the schematic. But it might also happen when there are bused ports on the schematic.

To determine the current value of this variable, type **printvar gen_create_netlist_busses**. For a list of all **schematic** variables and their current values, type **print_variable_group schematic**.

SEE ALSO

gen_dont_show_single_bit_busses

Controls whether single-bit buses are generated in the schematic.

TYPE

Boolean

DEFAULT

false

GROUP

schematic_variables

DESCRIPTION

This variable is used in conjunction with the **gen_show_created_busses** variable. When **gen_show_created_busses** is true and **gen_dont_show_single_bit_busses** is also set to true, single bit buses created by gen are not printed out. This suppresses messages about creation of single bit buses in the schematic.

To determine the current value of this variable use **printvar gen_dont_show_single_bit_busses**. For a list of all **schematic** variables and their current values, use the **print_variable_group schematic** command.

SEE ALSO

create_bus(2)

gen_match_ripper_wire_widths

Controls whether the **create_schematic** command generates rippers whose width always equals the width of the ripped net.

TYPE

Boolean

DEFAULT

false

GROUP

schematic_variables

DESCRIPTION

Controls whether the **create_schematic** command generates rippers whose width always equals the width of the ripped net. The default is *false*. When set to *true*, any rippers whose wire ends connect to scalar nets are of unit width.

By default (*false*), the **create_schematic** command connects scalar nets to the wire ends of multibit rippers. In such a case, all of the concerned bits of the ripper are assumed to be shorted together and connected to that scalar net.

To determine the current value of this variable, type **printvar gen_match_ripper_wire_widths**. For a list of all **schematic** variables and their current values, type **print_variable_group schematic**.

SEE ALSO

schematic_variables(3)

gen_max_compound_name_length

Controls the maximum length of compound names of bus bundles (for the **create_schematic -sge** command).

TYPE

integer

DEFAULT

256

GROUP

schematic_variables

DESCRIPTION

Controls the maximum length of compound names of bus bundles (for the **create_schematic -sge** command). The default is 256, which is the maximum length supported by SGE. Any buses with names longer than this variable are decomposed into their individual members in the schematic. If any such buses connect to cells referencing library symbols, those library symbols are ignored and, instead, new gen-created symbols are used. Any buses these gen-created symbols have whose compound names exceed this length are blown up into their individual members.

To determine the current value of this variable, use **printvar gen_max_compound_name_length**. For a list of all **schematic** variables and their current values, use the **print_variable_group schematic** command.

SEE ALSO

schematic_variables(3)

gen_max_ports_on_symbol_side

Specifies the maximum allowed size of a symbol created by **create_schematic**.

TYPE

integer

DEFAULT

0

GROUP

schematic_variables

DESCRIPTION

Specifies the maximum allowed size of a symbol created by **create_schematic**. For example, if this variable is five, symbols with no more than five ports on any one side are created. If this variable is not set or is set to zero, all input ports are placed on the left side of the symbol and all inout and output ports are on the right.

To determine the current value of this variable use **printvar gen_max_ports_on_symbol_side**. For a list of all **schematic** variables and their current values, use the **print_variable_group schematic** command.

gen_open_name_postfix

Specifies the postfix to be used by **create_schematic -sge** when creating placeholder net names for unconnected pins.

TYPE

Boolean

DEFAULT

""

GROUP

schematic_variables

DESCRIPTION

Specifies the postfix to be used by **create_schematic -sge** when creating placeholder net names for unconnected pins. The default is "".

The format of the net names is "%s%d%s", where the first "%s" is replaced by the value of **gen_open_name_prefix**, the second "%s" is replaced by the value of **gen_open_name_postfix**, and the "%d" is replaced by an integer whose value is generated automatically by **create_schematic -sge**.

For example, if **gen_open_name_prefix** = "Open", and **gen_open_name_postfix** = "_net", then the names created by **create_schematic** would be "Open1_net", "Open2_net", and so on.

The default values for **gen_open_name_prefix** and for **gen_open_name_postfix** are "Open" and "", respectively, so the default names created by **create_schematic** are Open1", "Open2", and so on.

To determine the current value of this variable, type **printvar gen_open_name_postfix**. For a list of all **schematic** variables and their current values, type **print_variable_group schematic**.

SEE ALSO

gen_open_name_prefix(3)
schematic_variables(3)

gen_open_name_prefix

Specifies the prefix to be used by **create_schematic -sge** when creating placeholder net names for unconnected pins.

TYPE

string

DEFAULT

Open

GROUP

schematic_variables

DESCRIPTION

Specifies the prefix to be used by **create_schematic -sge** when creating placeholder net names for unconnected pins. The default is "Open".

The format of the net names is "%s%d%s", where the first "%s" is replaced by the value of **gen_open_name_prefix**, the second "%s" is replaced by the value of **gen_open_name_postfix**, and the "%d" is replaced by an integer whose value is generated automatically by **create_schematic -sge**.

For example, if **gen_open_name_prefix** = "Open", and **gen_open_name_postfix** = "_net", then the names created by **create_schematic** would be "Open1_net", "Open2_net", and so on.

The default values for **gen_open_name_prefix** and for **gen_open_name_postfix** are "Open" and "", respectively, so the default names created by **create_schematic** are Open1", "Open2", and so on.

To determine the current value of this variable, type **printvar gen_open_name_prefix**. For a list of all **schematic** variables and their current values, type **print_variable_group schematic**.

SEE ALSO

gen_open_name_postfix(3)
schematic_variables(3)

gen_show_created_busses

Controls whether a message is printed out every time a schematic bus is created from cell pins for which no equivalent net bus exists in the netlist.

TYPE

Boolean

DEFAULT

false

GROUP

schematic_variables

DESCRIPTION

When true, a message is printed out every time a schematic bus is created from cell pins for which no equivalent net bus exists in the netlist. The default is false.

To determine the current value of this variable, type **printvar gen_show_created_busses**. For a list of all **schematic** variables and their current values, type **print_variable_group schematic**.

SEE ALSO

schematic_variables(3)

gen_show_created_symbols

Controls whether **create_schematic** prints a warning message every time it generates a new symbol for a cell because an appropriate symbol could not be found in the symbol libraries.

TYPE

Boolean

DEFAULT

false

GROUP

schematic_variables

DESCRIPTION

Controls whether **create_schematic** prints a warning message every time it generates a new symbol for a cell because an appropriate symbol could not be found in the symbol libraries. The default is false.

To determine the current value of this variable, use **printvar gen_show_created_symbols**. For a list of all **schematic** variables and their current values, use the **print_variable_group schematic** command.

SEE ALSO

schematic_variables(3)

gen_single_osc_per_name

Controls whether more than one off-sheet connector with any particular name is drawn on any schematic sheet.

TYPE

Boolean

DEFAULT

false

GROUP

schematic_variables

DESCRIPTION

When this variable is set to true, only one off-sheet connector with any particular name is drawn on any schematic sheet. In cases where there could potentially be more than one off-sheet connector with the same name on any schematic page, only one connector is drawn and the others just have their net segments show up as unconnected stubs.

To determine the current value of this variable use **printvar gen_single_osc_per_name**. For a list of all **schematic** variables and their current values, use the **print_variable_group schematic** command.

SEE ALSO

schematic_variables(3)

generic_symbol_library

Specifies the generic symbol library used for schematics.

TYPE

string

DEFAULT

generic.sdb

GROUP

schematic_variables

DESCRIPTION

The **db** file that contains generic symbols, templates, and layers used for schematics.

To determine the current value of this variable use **printvar generic_symbol_library**. For a list of all **schematic** variables and their current values, use the **print_variable_group schematic** command.

hdl_keep_licenses

Controls whether HDL licenses that are checked out remain checked out throughout the dc_shell session or are released after use.

TYPE

Boolean

DEFAULT

true

GROUP

hdl_variables

DESCRIPTION

When true (the default value), hdl licenses that are checked out remain checked out throughout the dc_shell session. When this variable is false, hdl licenses are released during execution of **compile**, after **compile** has completed the subtasks that require the license.

To determine the current value of this variable, type **printvar hdl_keep_licenses**. For a list of all **hdl** variables and their current values, type **print_variable_group hdl**.

SEE ALSO

hdl_variables(3)

hdl_preferred_license

Selects an hdl license to check out, if none is currently checked out.

TYPE

string

DEFAULT

""

GROUP

hdl_variables

DESCRIPTION

Selects an hdl license to check out, if none is currently checked out. Allowed values are *vhdl*, which selects the VHDL-Compiler license; or *verilog*, which selects the HDL-Compiler license. The elaboration process requires an hdl license.

SEE ALSO

hdl_variables(3)

hdl_variables

SYNTAX

```
string bus_dimension_separator_style =   "]["
string bus_minus_style =                  "-%d"
string bus_naming_style =                "%s[%d]"
long design_filename_length = "0"
Boolean hdl_keep_licenses =     "true"
string hdl_preferred_license =           ""
Boolean hdlin_auto_save_templates = "false"
Boolean hdlin_check_input_netlist = "false"
Boolean hdlin_check_no_latch = "false"
Boolean hdlin_elab_errors_deep = "false"
Boolean hdlin_enable_assertions = "false"
Boolean hdlin_enable_configurations = "false"
Boolean hdlin_enable_rtldrc_info = "false"
Boolean hdlin_ff_always_async_set_reset = "true"
Boolean hdlin_ff_always_sync_set_reset = "false"
string hdlin_field_naming_style = ""
string hdlin_generate_naming_style = "%s_%d"
string hdlin_generate_separator_style = "_"
Boolean hdlin_ignore_textio_constructs = "true"
Boolean hdlin_infer_function_local_latches = "false"
string hdlin_infer_multibit = "default_none"
string hdlin_infer_mux = "default"
string hdlin_keep_signal_name = "all_driving"
Boolean hdlin_latch_always_async_set_reset = "false"
Boolean hdlin_module_arch_name_splitting = "false"
int hdlin_module_name_limit = 256
int hdlin_mux_oversize_ratio = 100
int hdlin_mux_size_limit = 32
int hdlin_mux_size_min = 2
int hdlin_mux_size_only = 1
int hdlin_optimize_pla_effort = 2
string hdlin_preserve_sequential = "none"
string hdlin_presto_cell_name_prefix = "C"
string hdlin_presto_net_name_prefix = "N"
Boolean hdlin_prohibit_nontri_multiple_drivers = "true"
string hdlin_reporting_level = "basic"
Boolean hdlin_shorten_long_module_name = "false"
Boolean hdlin_subprogram_default_values = "false"
int hdlin_sv_ieee_assignment_patterns = 1
string hdlin_sv_packages = "enable"
Boolean hdlin_sv_tokens = "false"
Boolean hdlin_upcase_names = "false"
Boolean hdlin_vhdl93_concat = "true"
Boolean hdlin_vhdl_87 = "false"
int hdlin_vrlg_std = 2001
int hdlin_while_loop_iterations = 1024
string hlo_resource_allocation =  "constraint_driven"
string hlo_resource_implementation =  "use_fastest"
string systemcout_levelize = "true"
string template_naming_style = "%s_%p"
```

string **template_parameter_style** = "%s%d"
string **template_separator_style** = "_"
Boolean **verilogout_equation** = "false"
Boolean **verilogout_higher_designs_first** = "FALSE"
Boolean **verilogout_ignore_case** = "false"
list **verilogout_include_files** = {}
Boolean **verilogout_no_tri** = "false"
Boolean **verilogout_single_bit** = "false"
string **verilogout_unconnected_prefix** = "SYNOPSYS_UNCONNECTED_"
string **xterm_executable** = "xterm"

DESCRIPTION

These variables affect the **read** and **write** commands for Verilog and VHDL.

For more information on Verilog and VHDL variables, refer to the *HDL Compiler for Verilog Reference* and *VHDL Compiler Reference* manuals.

For a list of these variables and their current values, type **print_variable_group hdl**. To view this manual page online, type **help hdl_variables**. To view an individual variable description, type **help *var***, where *var* is the variable name.

bus_dimension_separator_style

This variable affects the **read** command with the **verilog, vhdl or edif** format options. In conjunction with the **bus_naming_style** variable, **bus_dimension_separator_style** controls the naming of individual bit-blasted ports derived from multidimensional arrays. This variable also affects the naming of bit-blasted multidimensional instance arrays and bit-blasted multidimensional net arrays in exactly the same way for ports.

bus_minus_style

This variable affects the **read** command with **vhdl** format option. For this option, this variable controls the naming of individual members of bit-blasted port, instance, or net busses with negative indices.

bus_naming_style

This variable affects the **read**, **write**, and **create_schematic** commands. For the **read** command with the **vhdl** or **verilog** format options, this variable controls the naming of bit-blasted ports in **dc_shell** when they are created from multidimensional busses in the original source.
For the **write** command with the **edif** format option, this variable controls the naming of individual members of each bus in the edif output. Also, together with the **bus_range_separator_style** and **bus_multiple_separator_style variables, this variable controls how ranged names of busses are written to the edif output.**
For the **create_schematic** command, this variable controls the naming of bus rippers that rip off bits from bussed nets. Together with the **bus_range_separator_style** variable, it also controls the naming of bussed nets, bussed ports, and bussed rippers in the schematic. Works the same as **write -f edif**.

hdl_keep_licenses

When *true* (the default value), hdl licenses that are checked out remain checked out throughout the **dc_shell** session. When this variable is *false*, hdl licenses are released during execution of **compile**, after **compile** has completed the subtasks that require the license.

hdl_preferred_license
: Selects an hdl license to check out, if none is currently checked out. Allowed values are *vhdl*, which selects the VHDL-Compiler license, or *verilog*, which selects the HDL-Compiler license.

hdlin_auto_save_templates
: Controls whether HDL designs containing parameters are read in as templates. The default is *false*.

hdlin_check_input_netlist
: Instructs the Verilog netlist reader in dc_shell/psyn_shell to check for the names conflict. The default is *false*.

hdlin_check_no_latch
: When *true*, a warning message is issued if a latch is inferred from a design. This is useful for verifying that a combinational design does not contain memory components. The default is *false*.

hdlin_elab_errors_deep
: Allows the user to be in debug mode to clean RTL with elaboration errors, link errors and internal errors or not. The default is *false*.

hdlin_enable_assertions
: Control Presto HDL compiler's use of SystemVerilog Assertions. The default is *false*.

hdlin_enable_configurations
: Control the configuration support by Presto VHDL. The default is *false*.

hdlin_enable_rtldrc_info
: When *true*, RTL TestDRC filename and linenumber information is created for designs processed by subsequent dc_shell commands. When *false*, no RTL TestDRC information is created.

hdlin_ff_always_async_set_reset
: When *true* (the default value), the HDL Compiler checks and reports asynchronous set and reset conditions of flip-flops. Setting this variable to false disables this behavior. Set this variable to false if you: do not use asynchronous set or reset conditions; do not want asynchronous set or reset devices inferred; and/or want your design to be input faster.

hdlin_ff_always_sync_set_reset
: When *true*, for a design subsequently analyzed, every constant 0 loaded on a flip-flop under the clock event is used for synchronous reset, and every constant 1 loaded on a flip-flop under the clock event is used for synchronous set. The default is *false*.

hdlin_field_naming_style
: Defines the parts of the net names that Presto HDL Compiler generates corresponding to the fields in VHDL records or in SystemVerilog structs. By default, the **hdlin_field_naming_style** is derived from the **bus_naming_style** and **bus_dimension_separator_style**. The default is empty string "".

hdlin_generate_naming_style

Specifies the naming style for *generated* design instances in VHDL designs. The default is *%s_%d*.

hdlin_generate_separator_style

Specifies the separator string for instances generated in multiple-nested loops. The default is _.

hdlin_ignore_textio_constructs

Controls whether the two commands **read** and **analyze** should warn or error when encountering STD.TEXTIO constructs in VHDL code. The default is *true*.

hdlin_infer_function_local_latches

Controls whether the Presto HDL Compiler infers latches inside functions and tasks. The default is *false*.

hdlin_infer_multibit

Specifies inference of multibit components for an entire design. The allowed values for **hdlin_infer_multibit** are *default_all*, *default_none*, and *never*. The default is *default_none*.

hdlin_infer_mux

When *all*, HDL Compiler attempts to infer a MUX_OP for a signal/variable assigned in a *case* statement. When *none*, HDL Compiler does not attempt to infer any MUX_OPs for a Verilog/VHDL design. When *default* (the default value), HDL Compiler attempts to infer a MUX_OP for a signal/variable assigned in a *case* statement if the statement is in a process associated with the **infer_mux** attribute/directive.

hdlin_keep_signal_name

Determines whether HDL Compiler attempts to keep a signal name. The allowed values are **all**, **none**, **user**, **user_driving**, and **all_driving**. The default is *all_driving*.

hdlin_latch_always_async_set_reset

Uses, for asynchronous reset, every constant 0 loaded on a latch, and uses, for asynchronous set, every constant 1 loaded on a latch, for a design subsequently analyzed. The default is *false*.

hdlin_module_arch_name_splitting

Controls whether Presto HDL Compiler recognizes a special format of Verilog module names, which allows users to specify both a module and an implementation architecture. The default is *false*.

hdlin_module_name_limit

The length threshold for compressing elaborated module names when **hdlin_shorten_long_module_name** is true. The default is *256*.

hdlin_mux_oversize_ratio

An integer that prevents inference of a sparse multiplexor. When the ratio of MUX_OP data inputs to unique data inputs is above the **hdlin_mux_oversize_ratio**, a MUX_OP will not be inferred. The default ratio is 100.

hdlin_mux_size_limit

An integer that limits the number of inputs of an inferred multiplexer. The default is 32. HDL Compiler does not generate MUX_OPs for multiplexers specified with more than this maximum number of inputs. The size of a multiplexer can be impounded by nested **if** or **case** statements.

hdlin_mux_size_min

Sets the lower bound for the number of inputs required to infer a multiplexer. The default value is *2*.

hdlin_mux_size_only

Controls which MUX_OP cells receive the **size_only** attribute in Presto HDL Compiler. When *0*, no cells receive the **size_only** attribute. When *1*, MUX_OP cells that are generated with the RTL **infer_mux** pragma and that are on set/reset signals receive the **size_only** attribute. When *2*, all MUX_OP cells that are generated with the RTL **infer_mux** pragma receive the **size_only** attribute. When *3*, all MUX_OP cells on set/reset signals receive the **size_only** attribute. When *4*, all MUX_OP cells receive the **size_only** attribute. The default value is *1*.

hdlin_optimize_pla_effort

Controls the effort the Presto HDL Compiler puts into optimization of PLA like constant CASE statements. The allowed values are *1*, *2* and *3*. The default value is *2*.

hdlin_preserve_sequential

Controls whether the **elaborate** and **read** commands retain unloaded sequential cells in the design. The allowed values are *none* (or *false*), *all* (or *true*), *all+loop_variables* (or *true+loop_variables*), *ff*, *ff+loop_variables*, *latch*, and *latch+loop_variables*. The default value is *none*.

hdlin_presto_cell_name_prefix

Sets the internal cell name prefix for HDL Compiler. The default value is *C*.

hdlin_presto_net_name_prefix

Sets internal net name prefix for HDL Compiler. The default value is *N*.

hdlin_prohibit_nontri_multiple_drivers

Controls whether the HDL Compiler issues an error, or only a warning, when it finds multiple drivers of a net. The default is *true*.

hdlin_reporting_level

Determines whether HDL Compiler prints which information in the report. The allowed values are *none*, *basic*, *comprehensive*, and *verbose*. The default is *basic*.

hdlin_shorten_long_module_name

Controls whether HDL Compiler compresses long names of elaborated modules. The default is *false*.

hdlin_subprogram_default_values

Determines which value HDL Compiler will use as the default value for variables, 'LEFT of its type or 0s. When this variable is set to *true*, 'LEFT of the type of variable is used as its default value. If you set this variable to *false*, 0s are used. The default is *false*.

hdlin_sv_ieee_assignment_patterns

Controls the level of support for IEEE-1800 SystemVerilog assignment patterns. The allowed values are *0*, *1* and *2*. When *0*, the IEEE-1800 SystemVerilog assignment patterns are not supported. When *1*, the IEEE-1800 SystemVerilog assignment patterns are supported in the right side of all legal, synthesizable assignment-like contexts, except module and interface instantiations. When *2*, the IEEE-1800 SystemVerilog assignment patterns are supported in the right side of all legal, synthesizable assignment-like contexts, including module instantiations. The default value is *1*.

hdlin_sv_packages

Specifies whether and how System Verilog packages should be analyzed. The allowed values are *none*, *chain*, *dont_chain*, and *enable*. The default value is *enable*.

hdlin_sv_tokens

Specifies whether a tokens file should be written out during the analysis of SystemVerilog designs. The default is *false*.

hdlin_upcase_names

Controls whether identifiers in the Verilog source code are converted to uppercase letters or left in their original case. The default is *false*.

hdlin_vhdl93_concat

Controls the concatenation behavior the tool uses to conform to the VHDL '93 Standard or the VHDL '87 Standard. The default is *true*.

hdlin_vhdl_87

Controls whether VHDL Compiler (Presto) follows VHDL '93 Standard or VHDL '87 Standard. The default is *false*.

hdlin_vrlg_std

Controls whether Presto Verilog/SystemVerilog enforces Verilog 1995 or Verilog 2001 or Verilog/SystemVerilog 2005 or SystemVerilog 2009. The allowed values are *1995*, *2001*, *2005*, and *2009*. The default is *2001*.

hdlin_while_loop_iterations

Places an upper bound on the number of times a loop is unrolled (to prevent potential infinite loops). The default is *1024*.

hlo_resource_allocation

Sets the default resource sharing type if the **resource_allocation** attribute is not set. Allowed values are *constraint_driven*, *area_only*, *area_no_tree_balancing* and *none*.

hlo_resource_implementation

Sets the default implementation selection type if the **resource_implementation** attribute is not set. Allowed values are *constraint_driven*, *area_only*, and *use_fastest*.

systemcout_levelize

Levelizes and flattens the netlist (when true) and replaces standard DesignWare operations with simulatable SystemC, before writing out the netlist, during **write -f systemc** command activity.

template_naming_style
: One of three string variables that determine the naming conventions for parameterized modules (templates). Determines what character(s) will appear between the design name and the parameter name. The default is an underscore(_).

template_parameter_style
: One of three string variables that determine the naming conventions for parameterized modules (templates). Determines what character(s) will appear between the parameter name and its value. The default is null (no separation).

template_separator_style
: One of three string variables that determine the naming conventions for parameterized modules (templates). Determines what character(s) will appear between parameter names for templates that have more than one parameter. The default is an underscore (_).

verilogout_equation
: When *true*, writes Verilog "assign" statements (Boolean equations) for combinational gates, rather than gate instantiations.

verilogout_higher_designs_first
: When *true*, writes Verilog "modules" ordered so that higher level designs come before lower level designs as defined by the design hierarchy. Default is to write lower level designs first.

verilogout_ignore_case
: When *true*, case is not considered when identifiers are compared against Verilog reserved words.

verilogout_include_files
: **write -f verilog** writes an include statement with the value that you set in this variable.

verilogout_no_tri
: When *true*, three-state nets are declared as Verilog "wire" instead of "tri." This variable is useful in eliminating "assign" primitives and "tran" gates in the Verilog output.

verilogout_single_bit
: When *true*, does not output vectored ports in Verilog output. Instead, all vectors are written as single bits.

verilogout_unconnected_prefix
: When this variable is set, the verilog writer in dc_shell uses this name to create unconnected wire names. The general form of the is "SYNOSPYS_UNCONNECTED_%d".

xterm_executable
: Specifies the path to an xterm program that is spawned to run Synopsys analysis tools such as RTL Analyzer or BCView. If an xterm is not found in your path, you can set this variable to point to an xterm executable.

SEE ALSO

elaborate (2), **read** (2), **write** (2).

hdlin_auto_save_templates

Controls whether HDL designs containing parameters are read in as templates.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

This variable, when set to true, reads in HDL designs containing parameters as templates. HDL parameter files include Verilog modules with parameter declarations and VHDL entities with generic declarations.

When an HDL file is read in as a template, it can be manipulated with the **printvar -templates**, **remove_template**, and **elaborate** commands. The template can also be built automatically when instantiated from another HDL design.

When false (the default value), HDL designs are read in and built instantly. This can be overridden for a design with a template pseudo comment.

For Verilog, place the following comment anywhere in the module:

```
   // synopsys template
```

For VHDL, place the following comment in the entity after the **port** declaration:

```
   -- synopsys template
```

To determine the current value of this variable, use the **printvar hdlin_auto_save_templates** command. For a list of HDL variables and their current values, use the **print_variable_group hdl** command.

SEE ALSO

elaborate(2)
template_parameter_style(3)
template_separator_style(3)

hdlin_check_input_netlist

Instructs the Verilog netlist reader in dc_shell/psyn_shell to check for the names conflict.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

Instructs the Verilog netlist reader in dc_shell/psyn_shell to check for names conflict.

The purpose of this variable is to issue warning if there are names conflict in the input netlist. Currently, the dc_shell/psyn_shell environment can handle names conflict, even though this is not support by the Verilog language, therefore it will pass by the error.

To determine the current value of this variable, use **printvar hdlin_check_input_netlist**. For a list of all **hdl** variables and their current values, use the **print_variable_group hdl** command.

SEE ALSO

hdl_variables(3)

hdlin_check_no_latch

Controls whether a warning message is issued if a latch is inferred from a design.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

Controls whether a warning message is issued if a latch is inferred from a design. When true, the warning message is issued. This is useful for verifying that a combinational design does not contain memory components. The default is false.

To determine the current value of this variable, type **printvar hdlin_check_no_latch**. For a list of **hdl** variables and their current values, type **print_variable_group hdl**.

SEE ALSO

hdl_variables(3)

hdlin_elab_errors_deep

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

Allows the user to be in debug mode to clean RTL with elaboration errors, link errors and internal errors or not.

Presto HDL compiler elaborates a design in a top-down hierarchical order. By default, this variable is *false*, elaboration failure of a certain module will prohibit the elaboration of all the child modules. This behavior is safe but less informative.

When this variable is set to *true*, Presto continues to elaborate its child modules even if some module's elaboration fails. In this way, Presto can elaborate as many modules as possible in a single elaboration run. Thus more RTL code elaboration problems can be detected in a single run and it improves users' efficiency.

This command works for both elaborate and read command.

To benefit from this command, the designs must be free from syntax errors during analyze step. In elab_errors_deep mode, we will not create any designs after elaboration. So commands such as current_design or link will not work in this debug mode.

SEE ALSO

hdl_variables(3)

hdlin_enable_assertions

Control Presto HDL compiler's use of SystemVerilog Assertions.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

Setting **hdlin_enable_assertions** to *true* allows the **elaborate** and **read_sverilog** commands to process a synthesis-friendly subset of SystemVerilog RTL Assertions. Facts added via assertions can contribute to optimization by ruling out machine states that cannot arise during normal operation of the design. This switch controls actions taken only during **elaborate**-phase, including elaborations that occur during linkage or compilation.

To be activated, an assertion statement must belong to a narrow category of synthesizable deferred immediate assertions. It must have a dedicated statement label that gives it a unique, lexically scoped, name. And it must be **analyze**'d in **sverilog** format (or by **read_sverilog**).
• The asserted expression should compare only one variable or signal to a compile-time constant, or else

• It should make a $onehot or $onehot0 claim about a collection of signal wires.

Consult the SystemVerilog User Guide for the exact definition of the subset currently supported by this release. All labelled assertions within this subset survive analysis regardless of the enable_assertions switch setting.

To **confirm** that an assertion should be considered during optimization:

• Set **hdlin_enable_assertions true** when it is being elaborated, *and*

• Set the lexically scoped name of its statement label as an element of the **confirmed_SVA()** array variable in the dc_shell environment in which the assertion is elaborated; give it the value *true*.

When all the above conditions are met, the labeled and confirmed assertions encoded in the intermediate (or template) design files can begin to influence logic minimization done by Presto HDL compiler. Note that either linking or elaborating designs with unresolved template references can cause subdesign elaborations whose confirmed assertions may also be exploited by Presto's optimizations. Here is an

example that confirms three assertions named "assertion_label1:,
"global_assertion1", and "assumption_about_port":

```
        analyze -f sverilog my_file.sv
         # The following lines can be brought in via a source command
         set confirmed_SVA(module1.scope1.assertion_label1) true
         set confirmed_SVA(module1.global_assertion1) true
         set confirmed_SVA(module2.generated_scope[0].assumption_about_port) true

         set hdlin_enable_assertions true
         elaborate module2
```

SEE ALSO

elaborate(2)

hdlin_enable_configurations

Controls the configuration support by Presto VHDL.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

The **hdlin_enable_configurations** variable enables configuration support by Presto VHDL, when set to **true**. The default value is **false**.

To specify a configuration, use the **analyze** command. To elaborate an entity with a specific configuration, use the **elaborate** command with the configuration name as the design name. For example, if file.vhdl contains the configuration named *my_configuration*, read and elaborate the design as follows:

```
   prompt> set hdlin_enable_configurations true
   prompt analyze -f vhdl file.vhdl
   prompt> elaborate my_configuration
```

The **read** command ignores configurations.

Setting this switch to *true* entails more stringent link-time checks. If designs do not use configurations, it is advised not to turn on this switch.

For more information refer to the *HDL Compiler (Presto VHDL) Reference Manual*.

SEE ALSO

analyze(2)
elaborate(2)

hdlin_enable_rtldrc_info

Controls whether RTL TestDRC creates file name and line number information for HDL constructs and instances for designs processed by subsequent dc_shell commands.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

Controls whether RTL TestDRC creates file name and line number information for HDL constructs and instances for designs processed by subsequent dc_shell commands.

When set to *true*, information about file name and line number for HDL constructs and instances is created for designs processed by subsequent dc_shell commands. In the Synopsys RTL Test Design Rule Checking (TestDRC) tool, this information must be created for the designs in order to view the links between gtech and mapped designs and HDL code. This analysis information is created when you process designs using the following dc_shell command: **dft_drc**.

When set to *false* (the default), no file or line information is created.

SEE ALSO

hdlin_ff_always_async_set_reset

Controls whether HDL Compiler checks and reports asynchronous set and reset conditions of flip-flops.

TYPE

Boolean

DEFAULT

true

GROUP

hdl_variables

DESCRIPTION

This variable controls whether HDL Compiler checks and reports asynchronous set and reset conditions of flip-flops. When **true** (the default value), the compiler checks and reports asynchronous set and reset conditions of flip-flops. Setting this variable to **false** disables this behavior. Set this variable to **false** if you do not use asynchronous set or reset conditions, if you do not want asynchronous set or reset devices inferred, or if you want your design to be input faster.

The **async_set_reset** pragma can be used to overwrite this variable. See the *HDL Compiler (Presto Verilog) Reference Manual* or the *HDL Compiler (Presto VHDL) Reference Manual* for more details about the pragma.

To determine the current value of this variable, type **printvar hdlin_ff_always_async_set_reset**. For a list of HDL variables and their current values, type **print_variable_group hdl**.

SEE ALSO

hdl_variables(3)

hdlin_ff_always_sync_set_reset

Controls whether every constant 0 loaded on a flip-flop under the clock event is used for synchronous reset, and every constant 1 loaded on a flip-flop under the clock event is used for synchronous set.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

Controls whether every constant 0 loaded on a flip-flop under the clock event is used for synchronous reset, and every constant 1 loaded on a flip-flop under the clock event is used for synchronous set. A setting of true for a design subsequently analyzed results in this behavior. The default is false.

For more details, refer to the chapter "Register, Multibit, Multiplexer, and Three-State Inference" in the *HDL Compiler for Verilog Reference Manual* or the *VHDL Compiler Reference Manual*.

To determine the current value of this variable, type **printvar hdlin_ff_always_sync_set_reset**. For a list of hdl variables and their current values, type **print_variable_group hdl**.

SEE ALSO

hdl_variables(3)

hdlin_field_naming_style

Defines the parts of the net names that Presto HDL Compiler generates corresponding to the fields in VHDL records or in SystemVerilog structs.

TYPE

string

DEFAULT

""

GROUP

hdl_variables

DESCRIPTION

Defines the parts of the net names that Presto HDL Compiler generates corresponding to the fields in VHDL records and SystemVerilog structs. By default, the hdlin_field_naming_style is derived from the bus_naming_style and bus_dimension_separator_style. But when these are "%s[%d]" and "][" or are "%s<%d>" and "><", then it is possible to specify an independent hdlin_field_naming_style, such as "%s<%s>", "%s[%s]", or "%s.%s", in which the first %s stands for the name up to the field and the second %s stands for the field. The hdlin_field_naming_style must be of the form "%sX%s" or "%sX%sY", where X and Y are (possibly identical) non-whitespace characters.

SEE ALSO

hdl_variables(3)
bus_naming_style(3)

hdlin_generate_naming_style

Specifies the naming style for *generated* design instances in VHDL designs.

TYPE

string

DEFAULT

GROUP

hdl_variables

DESCRIPTION

This Presto-VHDL variable specifies the naming style for *generated* design instances. For example,

```
      gen: for i in 0 to 1 generate
         U: CELL port map(a(i), z(i));
      end generate gen;
```

If **hdlin_generate_naming_style** is set to *%s_%d* (the default), the example results in instances named U_0 and U_1.

To determine the current value of this variable, type **printvar hdlin_generate_naming_style**. For a list of **hdl** variables and their current values, type **print_variable_group hdl**.

SEE ALSO

hdl_variables(3)

hdlin_generate_separator_style

Specifies the separator string for instances generated in multiple-nested loops.

TYPE

string

DEFAULT

_

GROUP

hdl_variables

DESCRIPTION

Specifies the separator string for instances generated in multiple-nested loops. For example,

```
        L1: for I in 3 downto 0 generate
           L2: for J in I downto 0 generate
              U: MY_AND port map ( A, B, Z);
           end generate;
        end generate;
```

If **hdlin_generate_separator_style** is set to __ (the default is _), the example results in instances named U_3__3, U_3__2, U_3__1, U_3__0, U_2__2, U_2__1, U_2__0, U_1__1, U_1__0, U_0__0.

To determine the current value of this variable, type **printvar hdlin_generate_separator_style**. For a list of **hdl** variables and their current values, type **print_variable_group hdl**.

The value of this variable affects only designs read in VHDL format.

SEE ALSO

hdl_variables(3)

hdlin_ignore_textio_constructs

Controls whether the two commands **read** and **analyze** should warn or error when encountering STD.TEXTIO constructs in VHDL code.

TYPE

Boolean

DEFAULT

true

GROUP

hdl_variables

DESCRIPTION

Controls whether the two commands **read** and **analyze** should error or only warn when encountering STD.TEXTIO constructs in VHDL code. Constructs from the STD.TEXTIO package are not supported in synthesis. When this variable is set to *true*, a warning is issued, but otherwise the constructs are ignored. If this default value is overridden to false, then instead an error message is generated on such constructs.

SEE ALSO

analyze(2)
elaborate(2)
hdl_variables(3)

hdlin_infer_function_local_latches

Controls whether the Presto HDL Compiler infers latches inside functions and tasks.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

Controls whether the Presto HDL Compiler infers latches inside functions and tasks.

When the value of this variable is *true*, latches inside function and task bodies can be inferred, according to the same rules used in always blocks.

When the value is *false* (the default), no latches are inferred in functions or tasks.

SEE ALSO

hdl_variables(3)

hdlin_infer_multibit

Specifies inference of multibit components for an entire design.

TYPE

string

DEFAULT

default_none

GROUP

hdl_variables

DESCRIPTION

Specifies inference of multibit components for an entire design. The allowed values for **hdlin_infer_multibit** are *default_all*, *default_none*, and *never*.

The *never* setting prevents inference of multibit components from HDL, regardless of directives (Verilog) or attributes (VHDL).

The *default_all* setting infers multibit components on all bused registers, three-states, and MUX_OPs, except where directives or attributes indicate otherwise.

The *default_none* setting specifies that only attributes or directives are used to infer multibit components. This is the default value for the **hdlin_infer_multibit variable**.

For details, refer to the *HDL Compiler for Verilog Reference Manual* or the *VHDL Compiler Reference Manual*.

To determine the current value of this variable, type **printvar hdlin_infer_multibit**. For a list of hdl variables and their current values, type **print_variable_group hdl**.

SEE ALSO

hdl_variables(3)

hdlin_infer_mux

Determines whether and how HDL Compiler infers a MUX_OP.

TYPE

string

DEFAULT

default

GROUP

hdl_variables

DESCRIPTION

Determines whether and how HDL Compiler infers a MUX_OP. Allowed values for **hdlin_infer_mux** are *all*, *none*, and *default*. When *all* is the value, HDL Compiler attempts to infer a MUX_OP for a signal or variable assigned in a case or if statement and for a expression using the ?: operator. When *none* is the value, HDL Compiler does not attempt to infer any MUX_OPs for a Verilog or VHDL design. When *default* (the default value) is the value, HDL Compiler attempts to infer a MUX_OP for a case or if statement and for the ?: operator, when the statement is in a process associated with the **infer_mux** attribute or directive. A MUX_OP cannot be inferred if it violates the **hdlin_mux_size_limit** variable.

For details, refer to the *HDL Compiler for Verilog Reference Manual* or the *VHDL Compiler Reference Manual*.

To determine the current value of this variable, type **printvar hdlin_infer_mux**. For a list of hdl variables and their current values, type **print_variable_group hdl**.

SEE ALSO

hdl_variables(3)
hdlin_mux_size_limit(3)

hdlin_keep_signal_name

Determines whether HDL Compiler attempts to keep a signal name.

TYPE

string

DEFAULT

all_driving

GROUP

hdl_variables

DESCRIPTION

The **hdlin_keep_signal_name** variable is available only in HDL Compiler (Presto Verilog) and HDL Compiler (Presto VHDL). For this discussion, these tools will be referred to as Presto. The original HDL and VHDL compilers do not support **hdlin_keep_signal_name**.

The value of **hdlin_keep_signal_name** determines whether Presto attempts to keep a signal name. This includes preserving cells between the signal and the input or output port. The **hdlin_keep_signal_name** variable provides Presto with guideline information for keeping a signal name, but it is not intended to be a 100% guarantee for keeping a signal. In some Presto optimizations, such as removing redundant code, the signal name may be optimized away, even if the signal is associated with the **keep_signal_name** attribute or directive. The allowed values are **all**, **none**, **user**, **user_driving**, and **all_driving**, as described below. The **user** and **user_driving** values require the **keep_signal_name** directive. Only Presto Verilog supports this directive; Presto VHDL does not support it.

none

Presto does not attempt to keep any signals. This value overrides the **keep_signal_name** directive.

all

Presto attempts to preserve a signal if the signal is not removed by optimizations. Dangling and driving nets are considered. This value does not guarantee that a signal is kept. The **all** value generally preserves fewer signals than the **user** value. For example, if there are 3 sequential nets named *N1*, *N2*, and *N3*, the **all** value may keep only *N1*. Setting the **hdlin_keep_signal_name** to the value of **user** and setting the **keep_signal_name** directive on *N1*, *N2*, and *N3* generally keeps *N1*, *N2*, *N3*.
This value may cause the **check_design** command to issue LINT-2 and LINT-3 warnings:
Warning: In design '...', net '...' has no drivers.
Logic 0 assumed. (LINT-3)

```
Warning: In design '...', net '...' driven by pin '
(no pin) ' has no loads.  (LINT-2)
```

all_driving

Presto attempts to preserve a signal if the signal is not removed by optimizations and the signal is in an output path. Only driving nets are considered. This value does not guarantee that a signal is kept. For signals that are not in an input path (do not have drivers), Presto connects the input to ground, assuming logic 0 (ground) for the driver.

user

This value works with the **keep_signal_name** directive. Presto attempts to preserve a signal if the signal is not removed by optimizations and that signal is labeled with the **keep_signal_name** directive. Dangling and driving nets are considered. Although not guaranteed, Presto usually keeps the specified signal for this configuration.
The **keep_signal_name** directive is not supported in Presto VHDL.

user_driving

This value works with the **keep_signal_name** directive. Presto attempts to preserve a signal if the signal is not removed by optimizations, the signal is in an output path, and the signal is labeled with the **keep_signal_name** directive. Only driving nets are considered. Although not guaranteed, Presto usually keeps the specified signal for this configuration. For signals that are not in an input path (do not have drivers), Presto connects the input to ground, assuming logic 0 (ground) for the driver.
The **keep_signal_name** directive is not supported in Presto VHDL.

EXAMPLES

In Example 1, you use the **hdlin_keep_signal_name** variable and the **keep_signal_name** directive to control what signals Presto attempts to preserve. (The **keep_signal_name** directive is supported only in Presto Verilog. It is not supported in Presto VHDL).

```
Example 1 - Verilog

module test (a, b, T,  A, c);
input  [3:0]   a;
input  [7:0]   b;
input T;
input A;
output reg  [7:0] c;
wire d, e;
//synopsys async_set_reset "A"
//synopsys keep_signal_name "e"

assign e = ( a[3] & ~a[2] &  a[1] & ~a[0] );
assign d = ( a[3] & ~a[2] &  a[1] & ~a[0] );

always @(T or b or A or d)
if (A)
     c = 8'h0;
       else if (T & d)
     c = b;
 endmodule
```

The internal wire e is not kept if the line //synopsys keep_signal_name "e" is removed, and **hdlin_keep_signal_name** is either **none**, **user**, **user_driving**, or **all_driving**. However, e is kept if **hdlin_keep_signal_name** is **all**. Also, e is kept if **hdlin_keep_signal_name** is **user** and //synopsys keep_signal_name is attached to "e".

In Example 2, by default Presto attempts to preserve the test1 and test2 signals because they are in output paths. Note that the test2 signal is not in an input path; Presto does not try to keep the test3 signal because it is not in an output path. Presto optimizes away the *syn1* and *syn2* signals.

Example 2 - Verilog

```
 module test12 (in1, in2, in3, in4, out1,out2 );
   input  [3:0]   in1;
   input  [7:0]   in2;
   input in3;
   input in4;
   output reg  [7:0] out1,out2;

   wire test1,test2, test3, syn1,syn2;

 //synopsys async_set_reset "in4"

   assign test1 = ( in1[3] & ~in1[2] & in1[1] & ~in1[0] );
  //test1 signal is in an input and output path
   assign test2 = syn1+syn2;
  //test2 signal is in a output path, but it is not in an input path
   assign test3 = in1 + in2;
  //test3 signal is in an input path, but it is not in an output path
   always @(in3 or in2 or in4 or test1)
   out2 = test2+out1;
   always @(in3 or in2 or in4 or test1)
       if (in4)
         out1 = 8'h0;
       else
       if (in3 & test1)
        out1 =  in2;
 endmodule
```

One way to keep the test3 signal is to set **hdlin_keep_signal_name** to **user** and place the **keep_signal_name** directive on test3, as shown in Example 3.

Example 3 - Verilog

```
 module test12 (in1, in2, in3, in4, out1,out2 );
   input  [3:0]   in1;
   input  [7:0]   in2;
   input in3;
   input in4;
   output reg  [7:0] out1,out2;

   wire test1,test2, test3, syn1,syn2;
 //synopsys keep_signal_name "test3"

 //synopsys async_set_reset "in4"

     assign test1 = ( in1[3] & ~in1[2] & in1[1] & ~in1[0] );
    //test1 signal is in an input and output path
     assign test2 = syn1+syn2;
    //test2 signal is in a output path, but it is not in an input path
     assign test3 = in1 + in2;
    //test3 signal is in an input path, but it is not in an output path
     always @(in3 or in2 or in4 or test1)
     out2 = test2+out1;
     always @(in3 or in2 or in4 or test1)
         if (in4)
           out1 = 8'h0;
         else
         if (in3 & test1)
          out1 = in2;
 endmodule
```

Table 1 shows how the variable settings affect the preservation of signal test3, with and without the directive applied to it. Asterisks (*) indicate that Presto does not attempt to keep the signal and might remove it.

Table 1 Variable and Directive Matrix for Signal test3

hdlin_keep_signal_name =	all	none	user	user_driving	all_driving
keep_signal_name is not set on test3	attempts to keep	*	*	*	*
keep_signal_name is set on test3	attempts to keep	*	attempts to keep	*	*

Table 2 shows how the variable settings affect the preservation of signal test2, with and without the directive applied to it. Asterisks (*) indicate that Presto does not attempt to keep the signal and might remove it.

Table 2 Variable and Directive Matrix for Signal test2

hdlin_keep_signal_name =	all	none	user	user_driving	all_driving
keep_signal_name is not set on test2	attempts to keep	*	*	*	attempts to keep
keep_signal_name is set on test2	attempts to keep	*	attempts to keep	attempts to keep	attempts to keep

Table 3 shows how the variable settings affect the preservation of signal test1, with and without the directive applied to it. Asterisks (*) indicate that Presto does not attempt to keep the signal and might remove it.

Table 3 Variable and Directive Matrix for Signal test1

```
hdlin_keep_signal_name =  all     none   user     user_driving   all_driving
---------------------------------------------------------------------------
keep_signal_name is     attempts   *      *            *          attempts
not set on test1        to keep                                   to keep

keep_signal_name is     attempts   *     attempts   attempts      attempts
set on test1            to keep          to keep    to keep       to keep
---------------------------------------------------------------------------
```

To determine the current value of this variable, use the following command:

```
   prompt> printvar hdlin_keep_signal_name
```

Use the following command to get a list of HDL variables and their current values in DB (dcsh) mode :

```
   prompt> print_variable_group hdl
```

Use the following command to get a list of HDL variables and their current values in XG and DB (Tcl) mode:

```
   prompt> print_variable_group
```

SEE ALSO

hdl_variables(3)

hdlin_latch_always_async_set_reset

Uses, for asynchronous reset, every constant 0 loaded on a latch, and uses, for asynchronous set, every constant 1 loaded on a latch, for a design subsequently analyzed.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

Uses, for asynchronous reset, every constant 0 loaded on a latch, and uses, for asynchronous set, every constant 1 loaded on a latch, for a design subsequently analyzed.

The default is false.

For more details, refer to the chapter "Register, Multibit, Multiplexer, and Three-State Inference" in the *HDL Compiler for Verilog Reference Manual* or the *VHDL Compiler Reference Manual*.

To determine the current value of this variable, type **printvar hdlin_latch_always_async_set_reset**. For a list of hdl variables and their current values, type **print_variable_group hdl**.

SEE ALSO

hdl_variables(3)

hdlin_module_arch_name_splitting

Controls whether Presto HDL Compiler recognizes a special format of Verilog module names, which allows users to specify both a module and an implementation architecture.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

During an **analyze** or **read** command, when **hdlin_module_arch_name_splitting** is *true* and a Verilog module name contains the special separator string "__" (two underscores), Presto HDL Compiler interprets the module name as specifying both a module and a specific architecture or implementation for that module. The portion of the original module name before the separator string becomes the new module name, and the portion after the string becomes the architecture.

For example, a module named "mod__impl" would be renamed to "mod" and marked as having architecture "impl."

If **hdlin_module_arch_name_splitting** is set to *false* (the default), the renaming never occurs and modules always retain their original names. If the variable is set to *true*, the "__" is used to divide the module name into module and architecture as described above.

This capability exists to allow the creation of synthetic libraries using Verilog, with multiple architectures per module.

To determine the current value of this variable, type **printvar hdlin_module_arch_name_splitting**. For a list of **hdl** variables and their current values, type **print_variable_group hdl**.

SEE ALSO

hdl_variables(3)

hdlin_module_name_limit

The length threshold for compressing elaborated module names when hdlin_shorten_long_module_name is true.

TYPE

int

DEFAULT

256

GROUP

hdl_variables

DESCRIPTION

When *hdlin_shorten_long_module_name* is *true*, the *hdlin_module_name_limit* variable is the length threshold for for whether the Presto HDL Compiler will compress names for elaborated modules.

When *hdlin_shorten_long_module_name* is *true*, if the name of an elaborated module is longer than the value of *hdlin_module_name_limit* (default 256), then an alternative, compressed name will be used that is easier for downstream tools to handle. The initial part of the name is still recognizable, but the tail of the name is replaced with numbers.

SEE ALSO

hdl_variables(3)
hdlin_shorten_long_module_name(3)

hdlin_mux_oversize_ratio

Prevents inference of a sparse multiplexer, when the ratio of MUX_OP data inputs to unique data inputs is above the **hdlin_mux_oversize_ratio**.

TYPE

int

DEFAULT

100

GROUP

hdl_variables

DESCRIPTION

Prevents inference of a sparse multiplexer, when the ratio of MUX_OP data inputs to unique data inputs is above the **hdlin_mux_oversize_ratio**.

The default ratio is 100.

Here are examples of code that infer a sparse MUX_OP:

```
VHDL
----
case sel is
  when "0001" =>   z <= data(3);
  when "0010" =>   z <= data(2);
  when "0100" =>   z <= data(1);
  when "1000" =>   z <= data(0);
  when others =>   z <= 'X';
end case;

Verilog
-------
case (sel)
  4'b0001: z = data[3];
  4'b0010: z = data[2];
  4'b0100: z = data[1];
  4'b1000: z = data[0];
  default: z = 1'bx;
endcase
```

When a MUX_OP is inferred from each of the previous examples, the mux oversize ratio is calculated. These examples infer a 16-to-1 MUX_OP, with 4 unique data inputs. The oversize ratio is 4. Setting **hdlin_mux_oversize_ratio** to 3 prevents either example inferring a MUX_OP. Any setting of 4 or greater allows these examples to infer a MUX_OP.

To determine the current value of this variable, type **printvar hdlin_mux_oversize_ratio**. For a list of hdl variables and their current values, type **print_variable_group hdl**.

SEE ALSO

hdl_variables(3)
hdlin_infer_mux(3)
hdlin_mux_size_limit(3)

hdlin_mux_size_limit

Limits the number of inputs of an inferred multiplexer.

TYPE

int

DEFAULT

32

GROUP

hdl_variables

DESCRIPTION

Limits the number of inputs of an inferred multiplexer. The default is 32. HDL Compiler does not generate MUX_OPs for multiplexers specified with more than this maximum number of 32 inputs. The size of a multiplexer can be impounded by nested if or case statements.

The time taken to process a mux is proportional to **hdlin_mux_size_limit** squared. Thus, extreme care must be taken when setting the limit to a figure larger than 32. If the design appears to hang with a limit greater than 32, lower the limit and try again. There is an additional nonpractical internal limit of 1073741824 for the size of a inferred mux.

For details, refer to the *HDL Compiler for Verilog Reference Manual* or the *VHDL Compiler Reference Manual*.

To determine the current value of this variable, type **printvar hdlin_mux_size_limit**. For a list of hdl variables and their current values, type **print_variable_group hdl**.

SEE ALSO

hdl_variables(3)
hdlin_infer_mux(3)

hdlin_mux_size_min

Sets the lower bound for the number of inputs required to infer a multiplexer.

TYPE

integer

DEFAULT

2

GROUP

hdl_variables

DESCRIPTION

Sets the lower bound for the number of inputs required to infer a multiplexer. The default value is *2*. HDL Compiler does not generate MUX_OPs for multiplexers specified with less than this minimum number of inputs.

SEE ALSO

hdl_variables(3)
hdlin_infer_mux(3)
hdlin_mux_oversize_ratio(3)
hdlin_mux_size_limit(3)

hdlin_mux_size_only

Controls which MUX_OP cells receive the **size_only** attribute in Presto HDL Compiler.

TYPE

integer

DEFAULT

1

GROUP

hdl_variables

DESCRIPTION

To ensure that MUX_OP GTECH cells are mapped to MUX technology cells, you must apply a **size_only** attribute to the cells to prevent logic decomposition in later optimization steps. Beginning with the B-2008.09-SP3 release, you can control which MUX_OP cells receive the **size_only** attribute by using the **hdlin_mux_size_only** variable. The following options are valid for **hdlin_mux_size_only**:
0 Specifies that no cells receive the **size_only** attribute
1 Specifies that MUX_OP cells that are generated with the RTL **infer_mux** pragma and that are on set/reset signals receive the **size_only** attribute
2 Specifies that all MUX_OP cells that are generated with the RTL **infer_mux** pragma receive the **size_only** attribute
3 Specifies that all MUX_OP cells on set/reset signals receive the **size_only** attribute: for example, MUX_OP cells that are generated by the **hdlin_infer_mux** variable set to **all**
4 Specifies that all MUX_OP cells receive the size_only attribute: for example, MUX_OP cells that are generated by the **hdlin_infer_mux** variable set to **all**.

SEE ALSO

set_size_only(2)
hdlin_infer_mux(3)

hdlin_optimize_pla_effort

Controls the effort the Presto HDL Compiler puts into optimization of PLA like constant CASE statements

TYPE

integer

DEFAULT

2

GROUP

hdl_variables

DESCRIPTION

This variable controls the effort that the Presto HDL Compiler puts into optimization of constant CASE statements that are FULL and PARALLEL.

The default value of this variable is 2. Default value can be overridden With 1 or 3. When the default value is overridden with 1, optimization of the constant CASE statement may be less likely. If the default value is overridden with 3, Presto will put higher effort in optimization of the constant CASE statements.

hdlin_preserve_sequential

Controls whether the **elaborate** and **read** commands retain unloaded sequential cells in the design.

TYPE

string

DEFAULT

none

GROUP

hdl_variables

DESCRIPTION

Use this variable to instruct the **elaborate** command or the **read** command whether the command is to retain unloaded sequential cells in the design. The default value is **none**. The following values are allowed:

none or false
    No unloaded sequential cells are preserved.

all or true
    All unloaded sequential cells are preserved, excluding unloaded sequential cells that are used solely as loop variables.

all+loop_variables or true+loop_variables
    All unloaded sequential cells are preserved, including unloaded sequential cells that are used solely as loop variables.

ff
    Only flip-flop cells are preserved, excluding unloaded sequential cells that are used solely as loop variables.

ff+loop_variables
    Only flip-flop cells are preserved, including unloaded sequential cells that are used solely as loop variables.

latch
    Only unloaded latch cells are preserved, excluding unloaded sequential cells that are used solely as loop variables.

latch+loop_variables
    Only unloaded latch cells are preserved, including unloaded sequential cells that are used solely as loop variables.

When elaborating an RTL-level design description, HDL Compiler (Presto) infers a sequential cell (a latch or flip-flop) for any Verilog **reg** or VHDL **variable** objects that are conditionally assigned or are assigned under a clock edge. However, if the design never reads the values of these objects or never uses the values to compute any output of the design, the HDL Compiler (Presto) tool, by default, does not retain sequential cells for these objects. These sequential cells have no loads or there is no path from them to the outputs, and so the tool does not need them to implement the design.

You may want to preserve unnecessary sequential cells, however, to improve observability of the design. The **hdlin_preserve_sequential** variable controls which unloaded sequential cells remain in the design.

The **+loop_variables** option affects which sequential cells are inferred when the value of **hdlin_preserve_sequential** is not set as **false** or **none**, and the **+loop_variables** option is not specified, the tool treats index variables as special, and no sequential cells are inferred.

If you do not want variables treated in this special way, specify **+loop_variables** for the variable values.

In the following example module, the bits of **fail** would ordinarily not exist as sequential cells after elaboration of the design. However, note the following:

• If you set the value of **hdlin_preserve_sequential** as **all**, **true**, or **ff**, the tool infers sequential cells for **fail**, and, as the option **+loop_variables** is not specified, the tool treats **i** in the example as special, because **i** is used only as a loop variable. Therefore, the tool does not infer sequential cells for **i**.

• If you set the value of **hdlin_preserve_sequential** as **all+loop_variables**, **true+loop_variables**, or **ff+loop_variables**, the tool infers sequential cells for **fail**, and, as the option **+loop_variables** is specified, the tool treats **i** the same way as it treats all other variables and assigns **i** to sequential cells.

• If you set the value of **hdlin_preserve_sequential** as **none** (the default), **false**, or **latch**, the tool does not assign either **fail** or **i** to sequential cells in the design.

Example 1:

```
  module test (clk,rst,error);
    parameter N = 8;
    input clk,rst,error;
    reg   [N-1:0] fail;
    integer i, j;

    always @(posedge clk or posedge rst)
      if (rst)
        fail <= {N-1{1'b0}};
      else
        for ( i = 0; i < N; i = i + 1 )
          fail[i] <= fail[i] | error;
  endmodule // test
```

Table 1 shows how the variable settings affect the sequential cell inferring for **fail** and **i** in Example 1.

Asterisks (*) indicate that Presto infers the sequential cells.

Table 1 Variables and Sequential Cell Inferring Matrix for module test

hdlin_preserve_sequential	all/true/ff		none/false/latch	
option "+loop_variables" is specified	true	false	true	false
reg [7:0] fail	*	*		
reg [31:0] i	*			

Important: The process of elaboration might preserve unloaded sequential cells, but the cells are deleted by the **compile** command (by default). To preserve the cells throughout the compile process, set the value of the **compile_delete_unloaded_sequential_cells** variable as **false**.

If there are other variables in your design that cause sequential elements to be inferred, but that you do not want in the final design, specify the preservation of sequential cells on a variable-by-variable basis, using the Verilog **preserve_sequential** pragma in a declaration.

For more information on the **preserve_sequential** pragma, see the *HDL Compiler (Presto Verilog) Reference Manual* or the *HDL Compiler (Presto VHDL) Reference Manual*.

SEE ALSO

elaborate(2)
compile_delete_unloaded_sequential_cells(3)

hdlin_presto_cell_name_prefix

Sets the internal cell name prefix for Presto HDL Compiler.

TYPE

string

DEFAULT

C

GROUP

hdl_variables

DESCRIPTION

This variable sets the internal cell name prefix for HDL Compiler. Use the **hdlin_presto_cell_name_prefix** variable to specify the cell name prefix. If you do not specify the variable, HDL Compiler generates the cell name with a "C" prefix.

The cell name prefix is only applied to cells whose names are not determined by another HDL Compiler cell naming convention.

The default value for the **hdlin_presto_cell_name_prefix** variable is fb"C".

SEE ALSO

hdl_variables(3)

hdlin_presto_net_name_prefix

Sets internal net name prefix for Presto HDL Compiler.

TYPE

String

DEFAULT

N

GROUP

hdl_variables

DESCRIPTION

The net name generated by Synopsys DB always has prefix "n". The net name generated by Presto has prefix "N" by default. User can use this variable to set Presto net name prefix to avoid inconsistent net name. The default value for the **hdlin_presto_net_name_prefix** variable is *"N"*.

SEE ALSO

hdl_variables(3)

hdlin_prohibit_nontri_multiple_drivers

Controls whether the Presto HDL Compiler issues an error, or only a warning, when it finds multiple drivers of a net.

TYPE

Boolean

DEFAULT

true

GROUP

hdl_variables

DESCRIPTION

Controls whether the Presto HDL Compiler issues an error, or only a warning, when it finds multiple drivers of a net. The variable works for Verilog and SystemVerilog input files.

Multiple drivers occur when a "reg" variable is driven by more than one always block or a "wire" variable is driven by more than one continuous assignment or input port, and when all of the drivers are not tristates.

HDL Compiler, in releases prior to 2001.08, does not issue an error message for multiple-driver nets. In releases 2001.08 and later, HDL Compiler issues an error, by default. You can convert this error, ELAB-366, to a warning, ELAB-365, by setting this variable to *false*. However, this setting is provided primarily for backward compatibility, and is not recommended. Inspect such warnings carefully. Setting **hdlin_prohibit_nontri_multiple_drivers** to *false* can result in invalid designs.

In simulation, if a reg variable is driven by more than one always block, the definition depends on which block executed most recently. In synthesis, because all blocks are concurrently executing at all times, this behavior is not possible, and invalid designs can result if the drivers are shorted together. Therefore, HDL Compiler issues an error if any bits of any variable are driven by more than one process.

The Verilog standard prohibits a multiple-driven wire variable, although some simulators permit this behavior. HDL Compiler issues the ELAB-366 error message in this situation. If you want multiple-driven wire variables, set **hdlin_prohibit_nontri_multiple_drivers** to *false*. Note that this variable setting might cause the generation of invalid designs.

Multiple drivers are permitted on nets declared as "tri," although a warning is issued if all the drivers are not tristate devices.

To determine the current value of this variable, type **printvar**

hdlin_prohibit_nontri_multiple_drivers. For a list of all **hdl** variables and their current values, type **print_variable_group hdl**.

SEE ALSO

hdl_variables(3)

hdlin_reporting_level

Determines which information Presto HDL Compiler prints in the report.

TYPE

string

DEFAULT

basic

GROUP

hdl_variables

DESCRIPTION

This variable controls the amount of output information to be included in the Presto (elaboration) report.

The following information is under control:
floating_net_to_ground prints the report for floating net connects to ground. This variable is better used in conjunction with the **set hdlin_keep_signal_name user** command and is not guaranteed to report all nets. The **check_design** command is recommended for detecting unconnected pins and ports.
fsm prints the report for inferred state variables.
inferred_modules prints the report for inferred sequential elements.
mux_op prints the report for MUX_OPs.
syn_cell prints the report for synthetic cells.
tri_state prints the report for inferred tri-state elements.

The **hdlin_reporting_level** variable can be set to 4 base settings: **none**, **basic**, **comprehensive**, and **verbose**, as shown in the following table:

Table 1 Base Settings

Information included in report	none	basic	comprehensive	verbose
floating_net_to_ground	false	false	true	true
fsm	false	false	true	true
inferred_modules	false	true	true	verbose
mux_op	false	true	true	true
syn_cell	false	false	true	true
tri_state	false	true	true	true

In addition to the base settings above, you can also modify the base settings to have fine grain control of individual reports through either adding (+) or subtracting (-) specific report(s) from the base setting with the following keywords:

```
floating_net_to_ground
fsm
syn_cell
mux_op
inferred_modules
tri_state
```

EXAMPLES

The following example uses **comprehensive-fsm**:

```
set hdlin_reporting_level comprehensive-fsm
```

The generated report shows the following settings:

```
floating_net_to_ground  true
fsm                     false
inferred_modules        true
mux_op                  true
syn_cells               true
tri_states              true
```

The following example uses **verbose-mux_op-tri_state**:

```
set hdlin_reporting_level verbose-mux_op-tri_state
```

The generated report shows the following settings:

```
floating_net_to_groundi true
fsm                     true
inferred_modules        verbose
mux_op                  false
syn_cell                true
tri_statei              false
```

The following example shows two commands that generate equivalent reports:

```
set hdlin_reporting_level basic+floating_net_to_ground+syn_cell+syn_cell+fsm

set hdlin_reporting_level comprehensive
```

SEE ALSO

hdl_variables(3)

hdlin_shorten_long_module_name

Controls whether the Presto HDL Compiler will compress long names of elaborated modules.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

Controls whether the Presto HDL Compiler will compress long names for elaborated modules.

When the value of this variable is *true*, if the name of an elaborated module is longer than the value of *hdlin_module_name_limit* (default 256), then an alternative, compressed name will be used that is easier for downstream tools to handle. The initial part of the name is still recognizable, but the tail of the name is replaced with numbers.

When the value is *false* (the default), the names of elaborated modules are uncompressed, even if they exceed the length given by hdlin_module_name_limit.

SEE ALSO

hdlin_module_name_limit(3)
hdl_variables(3)

hdlin_subprogram_default_values

Determines which value the compiler will use as the default value for variables, 'LEFT of its type or 0s.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

Determines which value the compiler will use as the default value for variables, 'LEFT of its type or 0s.

When this variable is set to *true*, 'LEFT of the type of variable is used as its default value. If you set this variable to *false* (the default), 0s are used.

SEE ALSO

hdl_variables(3)

hdlin_sv_ieee_assignment_patterns

Controls the level of support for IEEE-1800 SystemVerilog assignment patterns.

TYPE

integer

DEFAULT

1

GROUP

hdl_variables

DESCRIPTION

This variable controls the level of support for IEEE-1800 SystemVerilog assignment patterns.

The value of this variable can be overridden to 0 or to 2. When the value of this variable is 0, the IEEE-1800 SystemVerilog assignment patterns are not supported. When the value of this variable is set to 1, the IEEE-1800 SystemVerilog assignment patterns are supported in the right side of all legal, synthesizable assignment-like contexts, except module and interface instantiations. When the value of this variable is set to 2, the IEEE-1800 SystemVerilog assignment patterns are supported in the right side of all legal, synthesizable assignment-like contexts, including module instantiations.

Values less than 0 are treated as 0. Values greater than 2 are treated as 2.

SEE ALSO

hdl_variables(3)

hdlin_sv_packages

Specifies whether and how System Verilog packages should be analyzed.

TYPE

string

DEFAULT

enable

GROUP

hdl_variables

DESCRIPTION

Specifies which, if any, semantics the analyze or read command should apply when a -*format sverilog* source file declares a package. The setting affects the analyze step of SystemVerilog package declarations or references; it has no effect during elaborate: all synthesizable package semantics (including VHDL packages) are supported by elaborate, link and compile. The allowed values for **hdlin_sv_packages** are *none*, *chain*, *dont_chain* and the recommended setting: *enable*.

The setting *none* prevents parsing of package declarations or references.

All other settings accept "packages" (declarations, references and imports) as specified in section 19.2 of the IEEE-1800-2005 System Verilog standard. The default (and recommended) setting for SV package users is *enable*, it provides a synthesizable subset of packages that is compatible with the current release of Synopsys' VCS and Formality products.

Although the SV package standard was approved after the first commercial implementations had been released, there is only one known dialect incompatibility. If you receive a VER-934 Informational message, you may need to choose a specific package definition to be compatible with your local simulation tool. The only difference is in how an *import* statement treats names imported into the topmost (global) scope of a package_declaration:

- The *dont_chain* setting prevents imported names from being re-exported to clients of the package being declared.

- The *chain* setting always re-exports names that are imported into the global scope of a package; they may all be imported by the intermediate package's clients. An imported name and its definition which are re-exported ("chained" in VCS parlance) will not collide or interfere with copies of themselves in those cases where several intermediate packages redistribute content they acquired from a common source package (provided they all acquire it from a compatible analyzed version of the same source file).

In both variants, an imported name can be used freely within a package declaration to implement the package’s exported content. The analyzed result, a file named <package_identifier>.pvk, always contains a full copy of all imported content. A package’s .pvk file can stand alone; it does not require its clients to access the .pvk files that supplied its imported ingredients (unlike source-level file inclusion).

The chaining issue only concerns whether imported names become explicitly visible to an intermediate package’s clients as do objects explicitly declared at the outermost level of the intermediate package. Because a wildcard "import intermediate_pkg::*;" encumbers all of the exportable names found in "intermediate_pkg", the chain/ dont_chain choice can alter the outcome of name resolutions when several packages are combined in a downstream client.

The setting of hdlin_sv_packages *at the time a package is analyzed* is compiled into a "visibility" property on the imported, global names in the resulting .pvk file. This is an independent property that can be different at each level of a supply chain; it is not an inherited property of the name itself. A VER-934 informational message always indicates how this property is being set for those names where it may eventually matter.

Synopsys feels that intermediate (redistribution) packages are a valuable means to manage the "greatest hits" likely to be found in large design shops. In future releases, the *enable* setting will continue to track developments as the definition of SV packages converges on a comprehensive standard. The settings *chain* and *dont_chain* will continue to implement the *import_statement* semantics specified above, one of which will eventually be ruled non-standard.

For details, refer to the *IEEE-1800-2005 System Verilog Reference Manual* or the *VCS Simulation Compiler Reference Manual*.

To determine the current value of this variable, type **printvar hdlin_sv_packages**. For a list of hdl variables and their current values, type **print_variable_group hdl**.

SEE ALSO

analyze(2)
hdl_variables(3)

hdlin_sv_tokens

Specifies whether a tokens file should be written out during the analysis of SystemVerilog designs.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

When *hdlin_sv_tokens* is turned on, the lexical tokens that are sent to the parser during analysis of a SystemVerilog design (such as with an *analyze -f sverilog* or *read_sverilog*. command) will also be saved to an output file in the work directory.

When using complex macros or nested conditional compilation directives it can sometimes be a big help to see what the actual RTL will be after preprocessing, for example, to understand a mysterious syntax error.

When *hdlin_sv_tokens* is overridden to true, an expanded version of the source files is written to an output file *tokens.n{n}.sv*, instead of just being sent to the parser. In the output file, conditional compilation directives will already have been taken into account and all macro invocations will already have been expanded. The first tokens file written out in a particular work directory will be *tokens.1.sv*, the next *tokens.2.sv*, the next *tokens.3.sv*, and so on.

Although the output file is legal SystemVerilog, it is formatted in a way that is intended to be conveniently human readable, by using standard *line* directives to indicate the original source files and line numbers. This makes it relatively easy to track down the source of an error using the file name and line number in the error message. For example, if the error message refers to line 321 of file *"my_file.sv"*, then look for *line 321 "my_file.sv"* in the output file.

The only comments preserved are those few which are sent to the parser, because they are in some sense more than mere comments, such as synthesis pragmas and embedded scripts.

If any of the source files are encrypted, then the output file is not created.

SEE ALSO

analyze(2)
read_sverilog(2)

hdl_variables(3)

hdlin_upcase_names

Controls whether identifiers in the Verilog source code are converted to uppercase letters or left in their original case.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

Controls whether identifiers in the Verilog source code are converted to uppercase letters or left in their original case.

The default setting of this variable is *false*, which means that all names are left the way they are.

Setting the variable to *true* causes conversion of all identifiers in the Verilog code (variables, ports, and so on) to uppercase letters.

hdlin_vhdl93_concat

Controls the concatenation behavior the tool uses to conform to the VHDL ’93 Standard or the VHDL ’87 Standard.

TYPE

Boolean

DEFAULT

true

GROUP

hdl_variables

DESCRIPTION

The **hdlin_vhdl93_concat** variable is designed to control the concatenation behavior the Presto VHDL Compiler uses to conform to the VHDL ’93 Standard or the VHDL ’87 Standard. If you set the value of this variable as true (the default), the tool follows the VHDL ’93 Standard. If you set the value as false, the tool follows the VHDL ’87 Standard.

SEE ALSO

hdl_variables(3)
hdlin_vhdl_87(3)

hdlin_vhdl_87

Controls whether VHDL Compiler (Presto) follows VHDL ’93 Standard or VHDL ’87 Standard.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

The value you define for the **hdlin_vhdl_87** variable determines the standard the VHDL Compiler (Presto) is to follow in your design.

If you set the value of **hdlin_vhdl_87** as false (the default), the compiler will use the VHDL ’93 standard. If you set the value of **hdlin_vhdl_87** as true, the compiler will use the VHDL ’87 standard.

Regardless of the setting of this variable, the concatenation behavior is controlled by the variable hdlin_vhdl93_concat, whose default value is true.

SEE ALSO

hdl_variables(3)
hdlin_vhdl93_concat(3)

hdlin_vrlg_std

Controls whether Presto Verilog/SystemVerilog enforces Verilog 1995, Verilog 2001, Verilog/SystemVerilog 2005, or SystemVerilog 2009.

TYPE

integer

DEFAULT

2001

GROUP

hdl_variables

DESCRIPTION

This variable controls whether Presto Verilog/SystemVerilog is to enforce Verilog 1995, Verilog 2001, Verilog 2005, or SystemVerilog 2009 (a standard that incorporates Verilog 2005 into SystemVerilog).

If this variable is set to the value of 2001 (the default), Verilog 2001 is enforced.

If the variable is set to 1995, Verilog 1995 is enforced.

If the variable is set to 2005, Verilog 2005 or SystemVerilog 2005 is enforced, depending on the value set with the **-format** (**verilog** or **sverilog**, respectively) option for the **read** or **analyze** command.

If the variable is set to 2009, SystemVerilog 2009 is enforced. The SystemVerilog 2009 standard merges two previous standards, Verilog 2005 and SystemVerilog 2005, which defined extensions to it. There is no "Verilog 2009", but the **-format verilog** option for the **analyze** or **read** command is handled the same way as **-format sverilog** for backward compatibility. Those two standards were designed to be used as one language, so merging the base Verilog language and the SystemVerilog extensions into a single standard provides all information regarding syntax and semantics in a single document. Additionally, there are many extensions beyond SystemVerilog 2005 in SystemVerilog 2009.

SEE ALSO

hdl_variables(3)

hdlin_while_loop_iterations

Places an upper bound on the number of times a loop is unrolled (to prevent potential infinite loops).

TYPE

string

DEFAULT

1024

GROUP

hdl_variables

DESCRIPTION

Places an upper bound on the number of times a loop is unrolled (to prevent potential infinite loops). Loop unrolling occurs until the loop terminates. If you know that your loop will execute more times than the limit allows and that your loop will terminate at some point, increase the value of this variable.

The default value of this variable is 1024.

To determine the current value of this variable, type **printvar hdlin_while_loop_iterations**. For a list of hdl variables and their current values, type **print_variable_group hdl**.

SEE ALSO

hdl_variables(3)

hdlout_internal_busses

Controls the way in which the **write -format verilog** command and the **write -format vhdl** command write out internal bused nets by parsing the names of the nets.

TYPE

Boolean

DEFAULT

false

GROUP

io_variables

DESCRIPTION

The **hdlout_internal_busses** variable, when set to the value of *true*, controls the way in which the **write -format verilog** command and the **write -format vhdl** command write out internal bused nets by parsing the names of the nets.

When writing out VHDL files, be sure to set the **vhdlout_single_bit** variable and the **vhdlout_preserve_hierarchical_types** variable to "user" or "vector." Set the **bus_inference_style** variable and the **bus_naming_style** variable to the naming style. For more information, see the man pages for the **bus_inference_style** and **bus_naming_style** variables.

When writing out Verilog files, set the **verilogout_single_bit** variable to *false* (the default). Set the **bus_inference_style** variable and the **bus_naming_style** variable to the naming style. For more information, see the man pages for the **bus_inference_style** and **bus_naming_style** variables.

To determine the current value of this variable, type **printvar hdlout_internal_busses**. For a list of all **io** variables and their current values, type **print_variable_group io**.

SEE ALSO

bus_inference_style(3)
bus_naming_style(3)

hier_dont_trace_ungroup

Disables ungroup tracing set on the design with the **ungroup** command.

TYPE

Boolean

DEFAULT

0

DESCRIPTION

Disables ungroup tracing set on the design with the **ungroup** command. When **hier_dont_trace_ungroup** is set to 0 (the default), the **ungroup** command places on the design being ungrouped a string attribute that describes the ungroup operation. Other tools (for example, RTL Analyzer) can later use the attribute to recreate the ungroup operation and thus trace between the mapped and GTECH (generic) circuits.

Setting **hier_dont_trace_ungroup** to 1 disables ungroup tracing and can increase the efficiency of the **ungroup** and other commands.

SEE ALSO

ungroup(2)

high_fanout_net_pin_capacitance

Specifies the pin capacitance used to compute the loading of high-fanout nets.

TYPE

float

DEFAULT

1.000000

GROUP

timing_variables

DESCRIPTION

Specifies the pin capacitance used to compute the loading of high-fanout nets.

The pin capacitance for high-fanout nets is computed by multiplying the capacitance specified by the **high_fanout_net_pin_capacitance** variable times the high-fanout threshold.

For best results the pin capacitance chosen should be large enough to cause violations on all constrained high-fanout nets. This should force the nets to be replaced with buffer trees during compilation.

To determine the current value of this variable, type **printvar high_fanout_net_pin_capacitance**. For a list of all timing variables and their current values, type **print_variable_group timing**.

SEE ALSO

high_fanout_net_threshold(3)

high_fanout_net_threshold

Specifies the minimum number of loads for a net to be classified as a high-fanout net.

TYPE

integer

DEFAULT

1000

GROUP

timing_variables

DESCRIPTION

Specifies the minimum number of loads for a net to be classified as a high-fanout net.

Delays and loads of high-fanout are computed using a simplified model assuming a fixed fanout number. The rationale behind this is that delays of high-fanout nets are expensive to compute but such nets are often unconstrained (as in the case of global reset nets, scan enable nets, and so on). Those high-fanout nets that actually are constrained should eventually be replaced by buffer trees. So detailed delay calculations on such nets are expensive and usually unnecessary.

Setting the threshold to 0 (or to a very large number) ensures that no nets will be treated as high-fanout nets. However you should be aware that forcing fully accurate delay calculations on high-fanout can significantly increase compilation runtime in some cases.

The pin capacitance for high-fanout nets is computed by multiplying the capacitance specified by the **high_fanout_net_pin_capacitance** variable times the high-fanout threshold plus the number of net drivers.

The simplified net delay model is only used when computing data delays. Propagated clock latencies are always computed using the full accuracy net delay model.

To determine the current value of this variable, type **printvar high_fanout_net_threshold**. For a list of all timing variables and their current values, type **print_variable_group timing**.

SEE ALSO

high_fanout_net_pin_capacitance(3)

hlo_resource_allocation

Sets the default resource sharing type to be used by the **compile** command, if the **resource_allocation** attribute is not set.

TYPE

string

DEFAULT

constraint_driven

GROUP

hdl_variables

DESCRIPTION

Sets the default resource sharing type to be used by the **compile** command, if the **resource_allocation** attribute is not set. This variable has no effect in **compile_ultra**.

Allowed values are as follows:

constraint_driven (the default)
: Directs **compile** to share resources so that the timing constraints are met, or not made worse, by sharing.

area_only
: Directs **compile** to share operators without regard for timing constraints. All arithmetic expression trees are balanced.

area_no_tree_balancing
: Directs **compile** to share operators without regard to timing constraints. Arithmetic expression trees are not balanced.

none
: Directs **compile** to do no resource sharing, so that each operation is implemented with separate circuitry.

true
: Equivalent to *constraint_driven*; provided for backward compatibility with v2.0.

false
: Equivalent to *none*; provided for backward compatibility with v2.0.

You can override the value of **hlo_resource_allocation** for the current design by using the **set_resource_allocation** command to set the **resource_allocation** attribute.

To determine the current value of this variable, type **printvar hlo_resource_allocation**. For a list of all **hdl** variables and their current values, type **print_variable_group hdl**.

SEE ALSO

compile(2)
set_resource_allocation(2)
design_attributes(3)
hdl_variables(3)
synthetic_library(3)

ilm_enable_power_calculation

Perform power calculation on design which is to be used as ILM block.

TYPE

Boolean

DEFAULT

true

GROUP

power_variables

DESCRIPTION

create_ilm command processes the hierarchical block and remove the core logic, leaving only the interface logic behind. Subsequent power reporting commands cannot report power consumption correctly as some logic is missing in the netlist.

The **ilm_enable_power_calculation** variable is to instruct the **create_ilm** command to invoke power calculation API. Power consumption of the ILM design is calculated before core logic removal during **create_ilm**. The power data are stored as attributes on the ILM design, they will be used by **report_power** command during the final assembly step of the entire chip. Thus **report_power** command will be able to report more accurate power consumption for the designs with ILM blocks.

ilm_enable_power_calculation is on by default. If power license is not available, power calculation step will not be invoked.

ilm_ignore_percentage

Specifies a threshold for the percentage of total registers in the transitive fanout of an input port, beyond which the port is to be ignored when identifying interface logic.

TYPE

float

DEFAULT

25

DESCRIPTION

The value you specify for the **ilm_ignore_percentage** variable defines a minimum threshold for the percentage of the total registers in the transitive fanout of an input port, beyond which the tool is to ignore the port when identifying interface logic. The default is 25.

This variable affects the **-auto_ignore** option of the **create_ilm** command. The **-auto_ignore** option determines automatically those ports (for example, scan enable and reset ports) whose fanout should be ignored when **create_ilm** identifies objects as part of the interface logic model (ILM) for the design. The tool automatically ignores ports if the ports fan out to a percentage of total registers in the design that is greater than or equal to the value you specify for this variable.

For a discussion of ILM creation and the associated commands, see the manual page for the **create_ilm** command.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar ilm_ignore_percentage
```

SEE ALSO

create_ilm(2)

ilm_preserve_core_constraints

Enables the ILM mode (ilm_mode), which preserves constraints set on an interface logic model (ILM) core, if set to true.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

ILM mode supports budgeting of designs containing interface logic models (ILMs). When you set the value of the **ilm_preserve_core_constraints** variable as true, you enable the tool to characterize such designs *before* the creation of constraint scripts using the **write_script** command. For more information, see the man page for the **create_ilm** command.

Operating in ILM mode, the tool captures commands that set constraints, if the commands try to access design objects that are cut out of the design, that is, objects that are not part of the interface logic. The tool captures all commands that attempt to retrieve objects from within an ILM and that try to set constraints on the objects.

When you then invoke the **characterize** command on a cell (a design instance), the ILM mode attaches to the design all captured commands that could affect this instance or any module in the hierarchical subtree of this instance. This occurs *only* if the characterized cell is an ILM cell or if it contains an ILM in its subtree.

When you follow the characterization of a cell with the **write_script** command or the **write_environment** command on the design (still in ILM mode), the "attached" commands appear at the end of the **write_script** or **write_environment** output. The tool modifies the commands before they print out, as follows:

- Hierarchical paths in the arguments are modified.

- Variable references are substituted with the variable value.

In the generated script

- The current design is set to the module the script is generated for. This might be different from the current design at the time the captured command executed. This change of scope needs to be reflected in the command’s arguments

- Any variable reference ($<varname>) in the command line is replaced by the value of the corresponding variable at the time the captured command was executed.

Certain limitations to this approach exist, as follows:

• Use only very simple constraint commands. Do not use for loops, if statements, or any other Tcl construct.

• Use only the following commands while ILM mode is enabled:

concat
create_clock
find
list
remove_attribute
set_attribute
set_critical_range
set_disable_timing
set_dont_touch
set_dont_touch_network
set_false_path
set_input_delay
set_max_delay
set_min_delay
set_multicycle_path
set_operating_conditions
set_output_delay
set_wire_load_model

• You can also use commands starting with "get_", such as

get_cells
get_ports
and so on

and commands starting with "set_default_", such as

set_default_input_delay
set_default_load
and so on

• Change the value of the **ilm_preserve_core_constraints** variable only in the global scope. For example, do not enable ILM mode inside a for loop or inside a Tcl process.

• Do not change the value of Tcl variables while ILM mode is enabled.

• Do not use multiple commands in one line, separated by a semicolon.

• Do not source byte-code compiled Tcl scripts.

A command that the tool does not support will execute at the time you enter it at the command line (or source it). But the tool does not capture it, and therefore it is not added by **characterize**, nor is it printed out as a result of running **write_script**.

ILM mode is helpful in the following activities:

• When you set constraints on a design containing ILMs.

• When you perform budgeting and modifications on the design.

• When you create constraint files for subpartitions.

If you then load these subpartitions separately and replace the ILMs with the full design representation, the scripts ILM mode generates transfer all constraints from the original constraint file to that subpartition. This occurs even though you use an ILM representation to divide the constraints.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar ilm_preserve_core_constraints
```

EXAMPLES

In the following example, the tool reads in and processes a design that has one ILM called I_ALU, which is instantiated from the top level of the design.

```
psyn_shell-xg-t> read_db RISC_CORE_ILM.db
psyn_shell-xg-t> set ilm [get_cell I_ALU]
psyn_shell-xg-t> set dt "mult"
psyn_shell-xg-t> set ilm_preserve_core_constraints true
psyn_shell-xg-t> set_dont_touch [get_cells $ilm/*/$dt]
psyn_shell-xg-t> set_dont_touch [get_cells I_DATA_PATH/*]
psyn_shell-xg-t> characterize I_ALU
psyn_shell-xg-t> write_environment -cell I_ALU
           -constraints -out I_ALU.con
```

The above example reads in a design that contains an ILM named I_ALU; enables ILM mode; and sets some constraints on the design.

The example then creates a constraint script for the subinstance I_ALU (by characterizing it and invoking the **write_environment** command). The ILM mode adds to the generated output file the line **set_dont_touch [get_cells */mult]**.

The first **set_dont_touch** command tries to retrieve cells from within the ILM. Because the command uses wildcards, it might set the don't touch attribute on fewer cells than exist in the ILM, in comparison with the execution on the full design. Thus the command is added to the script. If you then apply the script to the full design, it will rerun the command and set the constraint on all cells.

```
Before the tool added the command to the script, a variable reference was replaced
by the variable value. In addition, the hierarchical path to the characterized
module I_ALU was removed from any argument. For example,

The tool first transforms the argument $ilm/*/$dt
   to
 I_ALU/*/mult.

When it removes the path, the argument becomes
  */mult
```

SEE ALSO

```
characterize(2)
create_ilm(2)
write_environment(2)
write_script(2)
```

initial_target_library

Specifies the list of technology libraries of components to be used for the first part of leakage power optimization in **place_opt**.

TYPE

list

DEFAULT

""

GROUP

system_variables

DESCRIPTION

Leakage power can be optimized for multi-vth designs in two different flows in **place_opt**. One is to use all the target libraries throughout optimization. The other is to use a subset of target libraries in the first part of the optimization steps and all the libraries in the second part. **initial_target_library** specifies the list of technology libraries of components to be used for the first part of leakage power optimization **place_opt**.

To determine the current value of this variable, use **echo $initial_target_library**.

SEE ALSO

system_variables(3)

insert_dft_clean_up

Causes the **insert_dft** command to use area recovery techniques to reduce the amount of test point logic.

TYPE

string

DEFAULT

true

GROUP

insert_dft variables

DESCRIPTION

Causes the **insert_dft** command to use area recovery techniques to reduce the amount of test point logic. The default is true. When the value of this variable is set to false, the **insert_dft** command does not optimize the test point logic.

To determine the current value of this variable, type **printvar insert_dft_clean_up**. For a list of all **insert_dft** variables and their current values, type **print_variable_group insert_dft**.

SEE ALSO

insert_dft(2)
preview_dft(2)

insert_dft_variables

Variables that affect the **insert_dft** and **preview_dft** commands.

SYNTAX

string **insert_dft_clean_up** = "true"
string **test_point_keep_hierarchy** = false
int **compile_dont_use_dedicated_scanout** = 1
string **insert_test_design_naming_style** = "%s_test_%d"
boolean **insert_test_map_effort_enabled** = "TRUE"
string **test_clock_port_naming_style** = "test_c%s"
boolean **test_dedicated_subdesign_scan_outs** ="FALSE"
int **test_default_min_fault_coverage** = 95
boolean **test_disable_find_best_scan_out** = "FALSE"
boolean **test_dont_fix_constraint_violations** = "FALSE"
int **test_isolate_hier_scan_out** = 0
string **test_mode_port_inverted_naming_style** = "test_mode_i%s"
string **test_mode_port_naming_style** = "test_mode%s"
string **test_non_scan_clock_port_naming_style** = "test_nsc_%s"
string **test_scan_clock_a_port_naming_style** = "test_sca%s"
string **test_scan_clock_b_port_naming_style** = "test_scb%s"
string **test_scan_clock_port_naming_style** = "test_sc%s"
string **test_scan_enable_inverted_port_naming_style** = "test_sei%s"
string **test_scan_enable_port_naming_style** = "test_se%s"
string **test_scan_in_port_naming_style** = "test_si%s%s"
string **test_scan_out_port_naming_style** = "test_so%s%s"
string **test_mux_constant_so** = "FALSE"
string **test_mux_constant_si** = "FALSE"

DESCRIPTION

These variables directly affect the **insert_dft** and **preview_dft** commands. Defaults are shown above, under Syntax.

For a list of **insert_dft** variables and their current values, type **print_variable_group insert_dft**. To view this manual page online, type **help insert_dft_variables**. To view an individual variable description, type **help *var***, where *var* is the name of the variable.

insert_dft_clean_up
    When *true* (the default), **insert_dft** uses area recovery techniques to reduce the amount of test point logic. When *false*, **insert_dft** does not optimize the test point logic.

test_point_keep_hierarchy
    When *false* (the default), **insert_dft** synthesizes test points and ungroups the test point design. When *true*, **insert_dft** keeps each test point design in a separate level of hierarchy.

compile_dont_use_dedicated_scanout
    When *1* (the default), test-ready **compile** (**compile -scan**), and subsequent compiles, do not use a scan cell's dedicated scan-out pin for functional

connections. When *0*, **compile** can use dedicated scan-out pins for functional connections.

insert_test_map_effort_enabled

When *false*, disables the **-map_effort** option of **insert_scan** and **insert_dft**, thus disabling any potential use of sequential mapping-based scan selection. When *true* (the default), the **-map_effort** option is enabled. Set this variable to *false* if you want the pre-3.3a behavior of **insert_dft** with respect to selection of scan equivalents.

insert_test_design_naming_style

Specifies how **insert_scan** and **insert_dft** names new designs created during the addition of test circuitry. When **insert_scan** or **insert_dft** modifies a design by adding test circuitry, it creates the design with a new, unique name. The new name is derived from the original design name and the format specified by this variable.

test_clock_port_naming_style

Specifies the naming style used by **insert_scan** and **insert_dft** for global test signal ports created in designs during the addition of test circuitry. New ports are not added if suitable ports are identified with the **set_scan_signal** or **set_signal_type** command.

test_dedicated_subdesign_scan_outs

Constrains how **insert_scan** and **insert_dft** route scan chains. When *true*, makes DFT Compiler create dedicated scan-out ports on subdesigns. When *false* (the default), DFT Compiler uses existing subdesign ports where possible.

test_default_min_fault_coverage

Specifies the default desired minimum fault coverage percent for the current design for the partial scan test methodology when the **set_min_fault_coverage** command has not be used for the current design.

test_disable_find_best_scan_out

When *false* (the default), **insert_scan** and **insert_dft** select scan cell scan-out pins that have the greatest timing slack. You can disable this behavior by setting the variable *true*.

test_dont_fix_constraint_violations

When *false* (the default), **insert_scan** and **insert_dft** attempt to minimize performance constraint violations. You can disable this behavior by setting the variable **true**.

test_isolate_hier_scan_out

When *1*, **insert_dft** inserts logic that isolates scan connections at hierarchical boundaries during functional operation; this can reduce dynamic switching currents and output loading. When *0* (the default), no logic is inserted. The variable does not affect **insert_scan** commands.

test_mode_port_inverted_naming_style

Specifies the style used to name new type test_hold_logic_zero test mode signal ports. New ports are not added if suitable ports are identified using **set_dft_signal**.

test_mode_port_naming_style
Used the same as **test_mode_port_inverted_naming_style** .

test_non_scan_clock_port_naming_style
Specifies how to name ports created by clock gating for nonscan clocks. The %s is filled with a string identifying the original clock, its inversion (either "i" for inverted or "n" for not-inverted), and inactive level (either 0 or 1).

test_scan_clock_a_port_naming_style
Used the same as **test_clock_port_naming_style**.

test_scan_clock_b_port_naming_style
Used the same as **test_clock_port_naming_style** .

test_scan_clock_port_naming_style
Used the same as **test_clock_port_naming_style** .

test_scan_enable_inverted_port_naming_style
Used the same as **test_clock_port_naming_style** .

test_scan_enable_port_naming_style
Used the same as **test_clock_port_naming_style** .

test_scan_in_port_naming_style
Specifies the naming style used by **insert_dft** for serial test-signal ports created in designs during the addition of test circuitry. New ports are not added if suitable ports are identified with the **set_scan_signal** or **set_signal_type** command.

test_scan_out_port_naming_style
Used the same as **test_scan_in_port_naming_style** .

test_mux_constant_so
Specifies how scan insertion uses a port you declare as scan output when this one is tied high or to the ground in functional mode. When you set this variable to *false* (the default value), scan insertion ignores the tie-off logic and directly uses the port as a scan output. This might change the output of the design during functional mode. When you set this variable to *true*, scan insertion multiplexes the scan output signal with the constant logic, using the scan enable signal to control the multiplexer.

test_mux_constant_si
Specifies how scan insertion uses a port you declare as scan input when this one is tied high or to the ground in functional mode. When you set this variable to *false* (the default value), scan insertion ignores the tie-off logic and directly uses the port as a scan input. This might change the output of the design during functional mode. When you set this variable to *true*, scan insertion multiplexes the scan input signal with the constant logic, using the scan enable signal to control the multiplexer.

SEE ALSO

insert_dft (2), **preview_dft** (2); **compile_dont_use_dedicated_scan_port** (3),

insert_test_design_naming_style (3), **test_clock_port_naming_style** (3), **test_default_min_fault_coverage** (3), **test_mode_port_inverted_naming_style** (3), **test_mode_port_naming_style** (3), **test_isolate_hier_scan_out** (3), **test_non_scan_clock_port_naming_style** (3), **test_scan_clock_a_port_naming_style** (3), **test_scan_clock_b_port_naming_style** (3), **test_scan_clock_port_naming_style** (3), **test_scan_enable_inverted_port_naming_style** (3), **test_scan_in_port_naming_style** (3), **test_scan_out_port_naming_style** (3), **insert_test_map_effort_enabled** (3).

insert_test_design_naming_style

Specifies how the **insert_dft** command names new designs created during the addition of test circuitry.

TYPE

string

DEFAULT

GROUP

insert_dft_variables

DESCRIPTION

Specifies how the **insert_dft** command names new designs created during the addition of test circuitry. When **insert_dft** modifies a design by adding test circuitry, it creates the design with a new, unique name. The new name is derived from the original design name and the format specified by this variable.

This variable must contain only one %s (percent s) and %d (percent d) character sequence. The percent sign has special meaning in the formatting process. To use a percent sign in the design name, two are needed in the variable setting (%%).

When **insert_dft** generates a new design name, it replaces %s with the original design name and %d with an integer. The integer is one that ensures the new name is unique. A single percent sign is substituted for %%.

For example, if this variable is set to %s_test_%d, and the original design name is my_design, the new design name is:

```
my_design_test_1
```

If the **insert_dft** command is repeated (for example, with a different test methodology), the new design name is:

```
my_design_test_2
```

To determine the current value of this variable, type **printvar insert_test_design_naming_style**. For a list of **insert_dft** variables and their current values, type **print_variable_group insert_dft** command.

SEE ALSO

insert_dft(2)
insert_dft_variables(3)

insert_test_variables

This command has become obsolete with 2000.04 ralease, and has been replaced by **insert_dft_variables**.

io_variables

Variables that affect the **read_file**, **read_lib**, **write**, and **write_lib** commands.

SYNTAX

Boolean **bus_inference_descending_sort** = "true"
string **bus_inference_style** = ""
string **equationout_and_sign** = "*"
string **equationout_or_sign** = "+"
Boolean **equationout_postfix_negation** = "true"
Boolean **hdlout_internal_busses** = "false"
integer **libgen_max_difference** = "1"
Boolean **pla_read_create_flip_flop** = "false"
Boolean **read_db_lib_warnings** = false
list **read_name_mapping_nowarn_libraries** = {"lsi_10k"}
Boolean **read_translate_msff** = false
string **sdfin_fall_cell_delay_type** = "maximum"
string **sdfin_fall_net_delay_type** = "maximum"
float **sdfin_min_fall_cell_delay** = 0.000000
float **sdfin_min_fall_net_delay** = 0.000000
float **sdfin_min_rise_cell_delay** = 0.000000
float **sdfin_min_rise_net_delay** = 0.000000
string **sdfin_rise_cell_delay_type** = "maximum"
string **sdfin_rise_net_delay_type** = "maximum"
string **sdfin_top_instance_name** = ""
float **sdfout_min_fall_cell_delay** = 0.000000
float **sdfout_min_fall_net_delay** = 0.000000
float **sdfout_min_rise_cell_delay** = 0.000000
float **sdfout_min_rise_net_delay** = 0.000000
float **sdfout_time_scale** = 1.000000
string **sdfout_top_instance_name** = ""
Boolean **sdfout_write_to_output** = "false"
list **write_name_mapping_nowarn_libraries** = {"lsi_10k"}
Boolean **write_name_nets_same_as_ports** = "false"

DESCRIPTION

These variables directly affect the **read_file**, **read_lib**, **write**, and **write_lib** commands.

For more about Verilog and VHDL variables, see the *HDL Compiler for Verilog Reference*, *VHDL Compiler Reference*, respectively, and the hdl_variables man page.

For a list of **io** variables and their current values, type **print_variable_group io**. To view this manual page online, type **help io_variables**. To view an individual variable description, type **help *var***, where *var* is the name of the variable.

bus_inference_descending_sort

Affects the **read_file** command with all format options except **db**, **verilog**, and **vhdl**; primarily used with the **lsi** option (LSI/NDL format). When *true* (the default variable), members of the port bus will be sorted in descending order rather than in ascending order.

bus_inference_style
: Affects the **read_file** command with all format options except **db**, **verilog**, and **vhdl**; primarily used with the **lsi** option (LSI/NDL format). This variable determines the style used to name inferred busses.

equationout_and_sign
: Specifies the "and" sign to use when writing a design in equation format. It must be either "*" or "&". If you specify an invalid value, the default is used.

equationout_or_sign
: Specifies the "or" sign to use when writing a design in equation format. It must be either "+" or " | ". If you specify and invalid value, the default is used.

equationout_postfix_negation
: When *true* (the default value), the ‘ (apostrophe) is used as the negation operator when writing a design in equation format. When *false*, the prefix negation operator ! (exclamation point) is used.

hdlout_internal_busses
: Controls how the **write -format verilog** and **write -format vhdl** commands write out bused nets by parsing their names when set to *true*.

libgen_max_difference
: Specifies to **read_lib** the maximum number of differences to list between the v3.1 format description of a library cell and its statetable description. The default value, -1, allows all differences to be listed.

pla_read_create_flip_flop
: Affects **read_file -f pla**. When *true*, enables output register information in PLA files to be read in and stored. When *false* (the default value), only **D** flip-flop information is read in and output register declarations are ignored.

read_db_lib_warnings
: When *true*, indicates that warnings are to be printed while a technology db library is being read in with **read_file**. When *false* (the default), no warnings are given.

read_name_mapping_nowarn_libraries
: Specifies a list of libraries for which no warning messages are to be issued by **read_file -f edif -names_file** if the libraries are not found. The default is to issue warning messages for all libraries not found.

read_translate_msff
: When *true*, indicates that master-slave flip-flops are to be automatically translated to master-slave latches while a technology db library or a technology library is being read in. When *false* (the default), both master and slave remain flip-flops. Note that DFT Compiler requires this variable to be set to *true*.
: This variable is used while a technology db library is being read in when

read_file is executed; and while a technology library is being read in by Library Compiler when **read_lib** is executed. The technology db library is affected ONLY if the program reports that the db library is being updated and asks you to save the results. Library Compiler always follows this variable during processing.
Note that DFT Compiler requires this variable to be set to *true*.

sdfin_fall_cell_delay_type
Specifies the delay type for fall cell delays read from a timing file in S.D.F. format. Delays from timing files are annotated in Design Compiler with **read_timing** .

sdfin_fall_net_delay_type
Specifies the delay type for fall net delays read from a timing file in S.D.F. format. Delays from timing files are annotated in Design Compiler with **read_timing** .

sdfin_min_fall_cell_delay
Specifies the minimum fall cell delay read from a timing file in S.D.F. format. Delays from timing files are annotated in Design Compiler with **read_timing** .

sdfin_min_fall_net_delay
Specifies the minimum fall net delay read from a timing file in S.D.F. format. Delays from timing files are annotated in Design Compiler with **read_timing** .

sdfin_min_rise_cell_delay
Specifies the minimum rise cell delay read from a timing file in S.D.F. format. Delays from timing files are annotated in Design Compiler with **read_timing** .

sdfin_min_rise_net_delay
Specifies the minimum rise net delay read from a timing file in S.D.F. format. Delays from timing files are annotated in Design Compiler with **read_timing** .

sdfin_rise_cell_delay_type
Specifies the delay type for rise cell delays read from a timing file in S.D.F. format. Delays from timing files are annotated in Design Compiler with **read_timing** .

sdfin_rise_net_delay_type
Specifies the delay type for rise net delays read from a timing file in S.D.F. format. Delays from timing files are annotated in Design Compiler with **read_timing** .

sdfin_top_instance_name
Specifies the name prepended to all instance names in timing files in S.D.F. format. Delays from timing files are annotated in Design Compiler with **read_timing** .

sdfout_min_fall_cell_delay
Specifies the minimum fall cell delay written to a timing file in S.D.F. format. Delays from Design Compiler are written to timing files with **write_timing**.

sdfout_min_fall_net_delay
Specifies the minimum fall net delay written to a timing file in S.D.F. format. Delays from Design Compiler are written to timing files with **write_timing**.

sdfout_min_rise_cell_delay
Specifies the minimum rise cell delay written to a timing file in S.D.F. format. Delays from Design Compiler are written to timing files with **write_timing**.

sdfout_min_rise_net_delay
Specifies the minimum rise net delay written to a timing file in S.D.F. format. Delays from Design Compiler are written to timing files with **write_timing**.

sdfout_time_scale
Specifies the time scale of the delays written to timing files in S.D.F. format. Delays from Design Compiler are written to timing files with **write_timing**.

sdfout_top_instance_name
Specifies the name prepended to all instance names when writing timing files in S.D.F. format. Timing files are written with the command **write_timing**. By default, **write_timing** prepends no name to all cell instance names.

sdfout_write_to_output
When *true*,**write_timing -f sdf** will write interconnect delays between cells and top level output ports, and will also write output-to-output pin IOPATH statements for cells that contain output-to-output timing arcs. The default is *false*. This variable must be set before using **write_timing**.

write_name_mapping_nowarn_libraries
Specifies a list of libraries for which no warning messages are to be issued by **write -f edif -names_file** if the libraries are not found. The default is to issue warning messages for all libraries not found.

write_name_nets_same_as_ports
When *true*, nets connected to ports have the same names as the ports in the descriptions of designs written in EDIF, LSI, or TDL format. The default is *false*. Other nets can be renamed to avoid creating shorts. Net names in the design are unchanged.

SEE ALSO

read_file(2) read_lib(2) write(2) write_lib(2) edif_variables(3) hdl_variables(3) vhdlio_variables(3)

layer_attributes

Contains attributes related to layer.

DESCRIPTION

Contains attributes related to layer.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified object. Specified with **list_attribute -class layer -application**, the definition of attributes can be listed.

Layer Attributes

data_type_details
Specifies detail information of data types on a layer.
The data type of **data_type_details** is string.
This attribute is read-only.

data_types
Specifies data types on a layer.
The data type of **data_types** is string.
This attribute is read-only.

defaultWidth
Specifies the default width of any dimension of an object on a layer.
The data type of **defaultWidth** is float.
This attribute is read-only.

fatContactThreshold
Specifies the threshold for using a fat wire contact on a lyer instead of the default contact.
The data type of **fatContactThreshold** is float.
This attribute is read-only.

fatFatMinSpacing
Specifies the minimum distance required between wires on a layer when the widths of both wires are greater than or equal to **fatWireThreshold**.
The data type of **fatFatMinSpacing** is float.
This attribute is read-only.

fatThinMinSpacing
Specifies the minimum distance required between wires on a layer when the width of one of the wires is greater than or equal to **fatWireThreshold**.
The data type of **fatThinMinSpacing** is float.
This attribute is read-only.

fatWireThreshold
Specifies the threshold for using the fat wire spacing rule instead of the default spacing rule on a layer.
The data type of **fatWireThreshold** is float.
This attribute is read-only.

isDefaultLayer

Specifies the layer used for routing when there are multiple layers with the same **mask_name** value.
The data type of **isDefaultLayer** is integer.
This attribute is read-only.

is_routing_layer

Specifies whether layer is a routing layer.
The data type of **is_routing_layer** is boolean.
This attribute is read-only.

layerNumber

Defines the number that identifies a layer.
The data type of **layerNumber** is integer.
This attribute is read-only.

layer_number

Defines the number that identifies a layer.
The data type of **layer_number** is integer.
This attribute is read-only.

layer_type

Specifies type of a layer.
The data type of **layer_type** is string.
This attribute is read-only.

mask_name

Specifies the physical layer associated with the specified layer object.
The data type of **mask_name** is string.
This attribute is read-only.

maxCurrDensity

Specifies the floating-point number representing in amperes per centimeter the maximum current density a layer can carry.
The data type of **maxCurrDensity** is double.
This attribute is read-only.

maxStackLevel

Defines the maximum number of vias that can stack at the same point.
The data type of **maxStackLevel** is integer.
This attribute is read-only.

minArea

Specifies the minimum area rule of any dimension of an object on a layer.
The data type of **minArea** is float.
This attribute is read-only.

minSpacing

Specifies the minimum separation distance between the edges of objects on a layer, if the objects are on different nets.
The data type of **minSpacing** is float.
This attribute is read-only.

minWidth

Specifies the minimum width of any dimension of an object on a layer.

The data type of **minWidth** is float.
This attribute is read-only.

name

Specifies name of a layer object.
The data type of **name** is string.
This attribute is read-only.

object_class

Specifies object class name of a layer, which is **layer**.
The data type of **object_class** is string.
This attribute is read-only.

pitch

Specifies the predominant separation distance between the centers of objects on a layer.
The data type of **pitch** is float.
This attribute is read-only.

preferred_direction

Specifies the preferred routing direction for a layer.
The data type of **preferred_direction** is string.
This attribute is read-only.

unitMaxCapacitance

Specifies the maximum capacitance of a layer.
The data type of **unitMaxCapacitance** is double.
This attribute is read-only.

unitMaxHeightFromSub

Specifies the maximum distance of a layer.
The data type of **unitMaxHeightFromSub** is double.
This attribute is read-only.

unitMaxResistance

Specifies the maximum resistance of a layer.
The data type of **unitMaxResistance** is double.
This attribute is read-only.

unitMaxSideWallCap

Specifies the maximum sidewall capacitance of a layer.
The data type of **unitMaxSideWallCap** is double.
This attribute is read-only.

unitMaxThickness

Specifies the maximum thickness of a layer.
The data type of **unitMaxThickness** is double.
This attribute is read-only.

unitMinCapacitance

Specifies the minimum capacitance of a layer.
The data type of **unitMinCapacitance** is double.
This attribute is read-only.

unitMinHeightFromSub
    Specifies the minimum distance of a layer.
    The data type of **unitMinHeightFromSub** is double.
    This attribute is read-only.

unitMinResistance
    Specifies the minimum resistance of a layer.
    The data type of **unitMinResistance** is double.
    This attribute is read-only.

unitMinSideWallCap
    Specifies the minimum sidewall capacitance of a layer.
    The data type of **unitMinSideWallCap** is double.
    This attribute is read-only.

unitMinThickness
    Specifies the minimum thickness of a layer.
    The data type of **unitMinThickness** is double.
    This attribute is read-only.

unitNomCapacitance
    Specifies the norminal capacitance of a layer.
    The data type of **unitNomCapacitance** is double.
    This attribute is read-only.

unitNomHeightFromSub
    Specifies the norminal distance of a layer.
    The data type of **unitNomHeightFromSub** is double.
    This attribute is read-only.

unitNomResistance
    Specifies the norminal resistance of layer.
    The data type of **unitNomResistance** is double.
    This attribute is read-only.

unitNomSideWallCap
    Specifies the norminal sidewall capacitance of a layer.
    The data type of **unitNomSideWallCap** is double.
    This attribute is read-only.

unitNomThickness
    Specifies the norminal thickness of a layer.
    The data type of **unitNomThickness** is double.
    This attribute is read-only.

visible
    Specifies a layer's visibility.
    The data type of **visible** is integer.
    This attribute is read-only.

SEE ALSO

get_attribute(2),
list_attribute(2),
report_attribute(2),

set_attribute(2).

lbo_cells_in_regions

Puts new cells at specific locations within a cluster.

TYPE

Boolean

DEFAULT

false

GROUP

links_to_layout_variables

DESCRIPTION

Puts new cells at specific locations within a cluster. Location based optimization (LBO) does this when the variable is set to *false* (the default). When set to *true*, LBO converts the specific location into a preferred region for the cell, by putting X_BOUNDS and Y_BOUNDS attributes on the cell when it is written to the PDEF file.

The proper setting for this variable depends on the engineering change order (ECO) capabilities of the back-end tools being used. Ideally the back-end tool will be able to support putting the new cells exactly where the **reoptimize_design** command wants them to go. If tools do not support that level of ECO, set this variable to *true* so that the PDEF file will at least contain regions into which the cells can be placed.

To determine the current value of this variable, type **printvar lbo_cells_in_regions**. For a list of all **links_to_layout** variables and their current values, type **print_variable_group links_to_layout**.

SEE ALSO

links_to_layout_variables(3)

level_shifter_naming_prefix

```
Using this variable, users can specify a prefix for the level shifters names
```

TYPE

```
string
```

DEFAULT

```
""
```

GROUP

```
mv
```

DESCRIPTION

```
When enable_special_level_shifter_naming variable is set to true, automatically
inserted level shifters by insert_level_shifters, compile and other commands are
named specially. The name follows the template <prefix> + <PD OR Design name> +
"_LS" + #. <prefix> is a user specifed prefix using the variable
level_shifter_naming_prefix. <PD OR Design Name> is the name of power domain where
the level shifter is being added, or the design name if the power domain is not
defined
```

SEE ALSO

```
enable_special_level_shifter_naming(3)
```

lib_thresholds_per_lib

Causes trip-point values in the Synopsys library to override user-specified values.

TYPE

Boolean

DEFAULT

true

GROUP

timing_variables

DESCRIPTION

Setting this variable as *false* causes user-specified trip-point values to override values defined in the Synopsys library. The default value is *true*.

Do not override the library trip-point values unless you are completely sure that these values are incorrect. Overriding trip-point values specified by the library creator is a questionable practice, which can result in inaccurate delay computations.

The following variables are affected:

rc_slew_lower_threshold_pct_fall rc_slew_lower_threshold_pct_rise rc_slew_upper_threshold_pct_fall rc_slew_upper_threshold_pct_rise rc_slew_derate_from_library rc_input_threshold_pct_rise rc_input_threshold_pct_fall rc_output_threshold_pct_rise rc_output_threshold_pct_fall

To determine the current value of this variable, type **printvar lib_thresholds_per_lib**.

For a list of all timing variables and their current values, type **print_variable_group timing**.

SEE ALSO

rc_input_threshold_pct_rise(3)
rc_input_threshold_pct_fall(3)
rc_output_threshold_pct_fall(3)
rc_output_threshold_pct_rise(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_rise(3)
rc_slew_derate_from_library(3)

lib_use_thresholds_per_pin

Causes pin specific trip-point values in the Synopsys library to override Library default trip-point values.

TYPE

Boolean

DEFAULT

true

GROUP

timing_variables

DESCRIPTION

Setting this variable as *false* causes Synopsys Library default trip-point values to override values defined for each library pin in the Synopsys library. The default value is *true*.

This variable is provided for backward compatibility. In case the user wants to use library defaults for all library pins instead of pin specific trip_point values, then this variable should be used.

The following variables are affected:

rc_slew_lower_threshold_pct_fall rc_slew_lower_threshold_pct_rise rc_slew_upper_threshold_pct_fall rc_slew_upper_threshold_pct_rise rc_slew_derate_from_library rc_input_threshold_pct_rise rc_input_threshold_pct_fall rc_output_threshold_pct_rise rc_output_threshold_pct_fall lib_thresholds_per_lib

To determine the current value of this variable, type **printvar lib_use_thresholds_per_pin**.

For a list of all timing variables and their current values, type **print_variable_group timing**.

SEE ALSO

rc_input_threshold_pct_rise(3)
rc_input_threshold_pct_fall(3)
rc_output_threshold_pct_fall(3)
rc_output_threshold_pct_rise(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_rise(3)
rc_slew_derate_from_library(3)

lib_thresholds_per_lib(3)

libgen_max_differences

Specifies to the **read_lib** command the maximum number of differences to list between the v3.1 format description of a library cell and its statetable description.

TYPE

integer

DEFAULT

-1

GROUP

io_variables

DESCRIPTION

Specifies to the **read_lib** command the maximum number of differences to list between the v3.1 format description of a library cell and its statetable description. The default value of -1 allows all differences to be listed.

For example, if **libgen_max_differences** = 5, **read_lib** lists only up to 5 differences between a library cell's v3.1 format description and its statetable description.

To see the value of this variable, type **printvar libgen_max_differences**. For a list of all **io** variables and their values, type **print_variable_group io**.

SEE ALSO

read_lib (2); **io_variables** (3).

library_attributes

Contains attributes placed on libraries.

DESCRIPTION

Contains attributes that can be placed on a library.

To set library attributes, use the **set_attribute** command. To determine the value of an attribute, use the **get_attribute** command. To remove attributes, use the **remove_attribute** command.

For more details on library attributes, see the *Library Compiler Reference Manual*. For information on all attributes, refer to the **attributes** manual page.

Library Attributes

default_fanout_load
    Fanout load of input pins in a library.

default_inout_pin_cap
    Capacitance of inout pins.

default_inout_pin_fall_res
    Fall resistance of inout pins.

default_inout_pin_rise_res
    Rise resistance of inout pins.

default_input_pin_cap
    Capacitance of input pins.

default_intrinsic_fall
    Intrinsic fall delay of timing arcs.

default_intrinsic_rise
    Intrinsic rise delay of a timing arc.

default_max_fanout
    Maximum fanout of pins.

default_max_transition
    Maximum transition of pins.

default_min_porosity
    Minimum porosity of designs.

default_output_pin_cap
    Capacitance of output pins.

default_output_pin_fall_res
    Fall resistance of output pins.

default_output_pin_rise_res
: Rise resistance of output pins.

default_slope_fall
: Fall sensitivity factor of a timing arc.

default_slope_rise
: Rise sensitivity factor of a timing arc.

k_process_drive_fall
: Process scale factor applied to the fall resistance of timing arcs.

k_process_drive_rise
: Process scale factor applied to the rise resistance of timing arcs.

k_process_intrinsic_fall
: Process scale factor applied to the intrinsic fall delay of timing arcs.

k_process_intrinsic_rise
: Process scale factor applied to the intrinsic rise delay of timing arcs.

k_process_pin_cap
: Process scale factor applied to pin capacitance of timing arcs.

k_process_slope_fall
: Process scale factor applied to the fall slope sensitivity of timing arcs.

k_process_slope_rise
: Process scale factor applied to the rise slope sensitivity of timing arcs.

k_process_wire_cap
: Process scale factor applied to the wire capacitance of timing arcs.

k_process_wire_res
: Process scale factor applied to the wire resistance of timing arcs.

k_temp_drive_fall
: Scale factor applied to timing arc fall resistance due to temperature variation.

k_temp_drive_rise
: Scale factor applied to timing arc rise resistance due to temperature variation.

k_temp_intrinsic_fall
: Scale factor applied to the intrinsic fall delay of a timing arc due to temperature variation.

k_temp_intrinsic_rise
: Scale factor applied to the intrinsic rise delay of a timing arc due to temperature variation.

k_temp_pin_cap
: Scale factor applied to pin capacitance due to temperature variation.

k_temp_slope_fall
: Scale factor applied to timing arc fall slope sensitivity due to temperature variation.

k_temp_slope_rise
: Scale factor applied to timing arc rise slope sensitivity due to temperature variation.

k_temp_wire_cap
: Scale factor applied to wire capacitance due to temperature variation.

k_temp_wire_res
: Scale factor applied to wire resistance due to temperature variation.

k_volt_drive_fall
: Scale factor applied to timing arc fall resistance due to voltage variation.

k_volt_drive_rise
: Scale factor applied to timing arc rise resistance due to voltage variation.

k_volt_intrinsic_fall
: Scale factor applied to the intrinsic fall delay of a timing arc due to voltage variation.

k_volt_intrinsic_rise
: Scale factor applied to the intrinsic rise delay of a timing arc due to voltage variation.

k_volt_pin_cap
: Scale factor applied to pin capacitance due to voltage variation.

k_volt_slope_fall
: Scale factor applied to timing arc fall slope sensitivity due to voltage variation.

k_volt_slope_rise
: Scale factor applied to timing arc rise slope sensitivity due to voltage variation.

k_volt_wire_cap
: Scale factor applied to wire capacitance due to voltage variation.

k_volt_wire_res
: Scale factor applied to wire resistance due to voltage variation.

nom_process
: Nominal process value used for library characterization. Fixed at 1.0 for most technology libraries.

nom_temperature
: Nominal ambient temperature used for library characterization. Usually 25 degrees Celsius. Multipliers use the nominal value to determine the change in temperature between nominal and operating conditions.

nom_voltage

Nominal source voltage value used in library element characterization. Typically 5 volts for a CMOS library. Multipliers use the nominal value to determine the change in voltage between nominal and operating conditions.

no_sequential_degenerates

When *true*, disables mapping to degenerated flip-flops or latches (that is, devices that have some input pins connected to 0 or to 1). The default for the attribute is *falseP or nonexistence, implying that degenerate devices will be allowed by default in the library. This attribute may be overridden on a component-by-component basis by using the attribute on library cells.*

sequential_bridging

When *true*, enables **Design Compiler** to take a multiplexed flip-flop and bridge (that is, connect) the output to the input to get a desired functionality. The default for the attribute is *false* or nonexistent, implying that bridging will be disabled by default in the library. Bridging is required for mapping in cases where there is no flip-flop with internal feedback in the target library but one is desired in the HDL. This attribute may be overridden on a component-by-component basis by using the attribute on library cells.
NOTE: Setting this attribute to *true* can result in an increase in run times and memory consumption for Design Compiler. The increased run times depend on the number of flip-flops in the target library or libraries which have the attribute set.

SEE ALSO

get_attribute(2)
remove_attribute(2)
set_attribute(2)
attributes(3)

library_cell_attributes

Contains attributes that can be placed on a library cell.

DESCRIPTION

Contains attributes that can be placed on a library cell.

To set an attribute, use the command identified in the individual description of that attribute. To determine the value of an attribute, use the **get_attribute** command. To remove attributes, use the **remove_attribute** command. If an attribute is "read-only," you cannot set it.

To determine the value of an attribute, use the **get_attribute** command. To remove attributes, use the **remove_attribute** command.

For a more detailed explanation of an attribute, refer to the manual pages of the appropriate **set** command. For information on all attributes, refer to the **attributes** manual page.

Library Cell Attributes

dont_touch
: Identifies library cells to be excluded from optimization. Values are *true* (the default) or *false*. Library cells with the **dont_touch** attribute set to *true* are not modified or replaced during **compile**. Setting **dont_touch** on a hierarchical cell sets the attribute on all cells below it. Set with **set_dont_touch**.

dont_use
: Disables the specified library cells so that they are not added to a design during **compile**. Set with **set_dont_use**.

no_sequential_degenerates
: When *true*, disables mapping to versions of this latch or flip flop that have some input pins connected to 0 or to 1. Set with **set_attribute**. This attribute may also be set on the library itself, and that value will apply as the default for all registers in the library which do not have the attribute set individually.

preferred
: Specifies the preferred library gate to use during technology translation when there are other gates with the same function in the target library. Set with **set_prefer**.

scan
: When *true*, specifies that the instances of the library cell are always replaced by equivalent scan cells during **insert_dft**. When *false*, instances are not replaced. Set with **set_scan**.

scan_group
: A user-defined string variable that allows you to specify to DFT Compiler a preferred scan equivalent for a non-scan storage element, when a library

contains multiple scan equivalents. Typical values are *low*, *medium*, and *high*, for low, medium and high drive strengths. However, you can define any string variable, and it need not describe drive strength. The default behavior is for DFT Compiler to attempt to choose a scan element that best matches the electrical characteristics of the non-scan element. For a more detailed explanation, refer to the *DFT Compiler Reference Manual*. The matching of electrical characteristics works well with the standard CMOS delay model, but is not accurate with other delay models. Normally, **scan_group** would be set by the ASIC vendor or library developer, but you can also set **scan_group**. Consult your ASIC vendor before attempting to set **scan_group** with **set_attribute**. For more information about **scan_group**, refer to the *DFT Compiler Reference Manual*.

sequential_bridging

When *true*, enables Design Compiler to take a multiplexed flip-flop and bridge (that is, connect) the output to the input to get a desired functionality. The default is *false*, so this attribute must be set in order to enable the functionality. Bridging is required for mapping in cases where there is no flip-flop with internal feedback in the target library but one is desired in the HDL. Set with **set_attribute**. This attribute may also be set on the library itself, and that value will apply as the default for all registers in the library which do not have the attribute set individually.
NOTE: Setting this attribute to *true* can result in an increase in run times and memory consumption for Design Compiler. The increased run times depend on the number of flip-flops in the target library or libraries for which this attribute has been set.

SEE ALSO

get_attribute(2)
remove_attribute(2)
set_attribute(2)
attributes(3)

link_force_case

Controls the case-sensitive or case-insensitive behavior of the **link** command.

TYPE

string

DEFAULT

check_reference

GROUP

system_variables

DESCRIPTION

Controls the case-sensitive or case-insensitive behavior of the **link** command. The value of this variable can be *case_sensitive*, *case_insensitive*, or *check_reference*. The default value is *check_reference*; and the reference being linked is checked to ascertain the case-sensitivity of the input format that created that reference. That case-sensitivity is then enforced. For example, a VHDL reference is linked case-insensitively, and a Verilog reference is linked case-sensitively. To override this behavior, you can set the value of the **link_force_case** variable to either *case_sensitive* or *case_insensitive*.

Setting this variable to *case_insensitive* is not recommended if you are reading in any source files from formats that are case-sensitive, such as Verilog. To do so can lead to inconsistent and undesirable results.

To determine the value of this variable, use **printvar link_force_case**. For a list of all **system** variables and their values, use the **print_variable_group system** command.

SEE ALSO

link(2)

link_library

Specifies the list of design files and libraries used during linking.

TYPE

list

DEFAULT

* your_library.db

GROUP

system_variables

DESCRIPTION

Specifies the list of design files and libraries used during linking. The **link** command looks at those files and tries to resolve references in the order of specified files. A "*" entry in the value of this variable indicates that the **link** command is to search all the designs loaded in dc_shell while trying to resolve references. If file names do not include directory names, files are searched for in the directories in **search_path**. The default is {"*" your_library.db}. Change *your_library.db* to reflect your library name.

To determine the current value of this variable, use **$list link_library**. For a list of all **system** variables and their current values, use the **print_variable_group system** command.

SEE ALSO

link(2)

links_to_layout_variables

Variables that affect the **links_to_layout** capability.

SYNTAX

```
Boolean auto_wire_load_selection = "true"
Boolean compile_disable_area_opt_during_inplace_opt = "false"
Boolean compile_dont_touch_annotated_cell_during_inplace_opt = "false"
Boolean compile_ignore_area_during_inplace_opt = "false"
Boolean compile_ignore_footprint_during_inplace_opt =  "false"
Boolean compile_ok_to_buffer_during_inplace_opt = "false"
Boolean compile_update_annotated_delays_during_inplace_opt =  "true"
Boolean lbo_buffer_insertion_enabled =  "false"
Boolean lbo_buffer_removal_enabled =  "false"
Boolean lbo_cells_in_regions =  "false"
Boolean ltl_enable_mean_physical_port_location = "false"
Boolean ltl_obstruction_type = "placement_only"
string  reoptimize_design_changed_list_file_name  =    ""
Boolean reoptimize_design_disable_area_opt_during_postlayout_opt = "false"
Boolean sdfout_allow_non_positive_constraints = "false"
```

DESCRIPTION

These variables directly affect the **links_to_layout** capability. Defaults are shown above, under Syntax.

For a list of these variables and their current values, type **print_variable_group links_to_layout**. To view this manual page online, type **help links_to_layout_variables**. To view an individual variable description, type **help *var***, where *var* is the variable name.

auto_wire_load_selection
When *true* (the default value), turns on the automatic selection of the wire load model.

compile_disable_area_opt_during_inplace_opt
When *true*, disables area optimization during inplace optimization. When *false* (the default value), area optimization is enabled; that is, **compile -in_place** attempts to save design area by downsizing cells that are not a part of the critical path. Note that this downsizing can sometimes help speed up the critical path. Thus, setting this option to *true*, while reducing the number of changes made to the design, may hinder this command from achieving the best timing optimization results.
Note: This variable no longer works for **reoptimize_design**.

compile_dont_touch_annotated_cell_during_inplace_opt
When *true*, **reoptimize_design -in_place** and **compile -in_place** disallow swapping cells that have annotated delays. When *false* (the default value), **reoptimize_design -in_place** and **compile -in_place** allow annotated cells to be swapped for cells without annotated delay.

compile_ignore_area_during_inplace_opt

When *true*, **compile -in_place** is allowed to swap cells that have the same number of pins, pin names, logic functionality, and footprint, regardless of area. When *false* (the default value), cells are not swapped if they have different areas.
Note: This variable no longer works for **reoptimize_design**.

compile_ignore_footprint_during_inplace_opt

When *true*, **compile -in_place** is allowed to swap cells that have the same number of pins, pin names, logic functionality, and cell area, regardless of footprint. When *false* (the default value), cells are not swapped if they have different footprints.
Note: This variable no longer works for **reoptimize_design**.

compile_ok_to_buffer_during_inplace_opt

When *true*, **compile -in_place** is allowed to add buffers (or two inverters in a row) to the design to help meet timing constraints and fix design rule violations, including minimum path timing violations. The default is *false*.
Note: This variable no longer works for **reoptimize_design**.

compile_update_annotated_delays_during_inplace_opt

When *true* (the default value), **reoptimize_design -in_place** and **compile -in_place** are allowed to modify the values of annotated delays on nets connected to the swapped cells and to remove annotated delays on cells connected to the swapped cells. When *false*, **reoptimize_design -in_place** and **compile -in_place** disallow annotated delays to be modified.

lbo_buffer_insertion_enabled

When *false* (the default value), location based optimization is not used when doing buffer insertion.

lbo_buffer_removal_enabled

When *false* (the default value), location based optimization is not used when doing buffer removal, and in fact, buffer removal itself is not done during IPO or PLO.

lbo_cells_in_regions

When *false* (the default value), location based optimization will insert new cells at specific locations within clusters. When *true*, new cells are instead inserted into 'preferred' regions via X_BOUNDS and Y_BOUNDS attributes on the cells in the PDEF file. The best setting of this variable is best on the ECO capabilities of the back-end P&R or floorplanning too.

ltl_enable_mean_physical_port_location

When *true*, **reoptimize_design** uses the arithmetic mean of all physical locations for ports that have multiple physical equivalents. When *false* (the default), **reoptimize_design** ignores nets of ports that have multiple physical equivalents.

ltl_obstruction_type

Controls the routing blockage type for the named obstructions, without route type being specified.

reoptimize_design_changed_list_file_name

The name of a file to which **reoptimize_design -in_place** and **reoptimize_design**

-post_layout_opto commands should output a list of design modifications/ additions.

reoptimize_design_disable_area_opt_during_postlayout_opt

When *false* (the default setting), **reoptimize_design -post_layout_opto** attempts to downsize cells that are not a part of the critical path, in order to save design area. This occurs after cell groups along the critical path have been restructured to meet timing constraints. Setting this variable to *true* will cause **reoptimize_design -post_layout_opto** to skip the area downsizing step. By skipping this step, the changes made by **reoptimize_design** will be restricted to just those that improve the timing of the critical path or the legality of the design (**max_transition** or **min_path**) violations).

sdfout_allow_non_positive_constraints

When *true*, **write_constraints -format sdf** can write out PATHCONSTRAINT constructs with nonpositive (<= 0) constraint values. When *false* (the default), paths with nonpositive constraints are written with a constraint value of 0.01.

SEE ALSO

compile (2), **help** (2) **list** (2) **reoptimize_design** (2), **set_wire_load** (2), **write_constraints** (2); **compile_variables** (3), **io_variables** (3).

ltl_obstruction_type

Controls the routing blockage type for the named obstructions, without route type being specified.

TYPE

Boolean

DEFAULT

placement_only

GROUP

links_to_layout_variables

DESCRIPTION

Controls the routing blockage type for the named obstructions, without route type being specified. When the setting is placement_only (the default), the obstructions are treated as placement obstruction only and routing wires can still go through. When the setting is routing_none, the obstructions are treated as routing blockages and no routing wires are allowed.

To determine the current value of this variable, type **printvar ltl_obstruction_type**. For a list of all links_to_layout variables and their current values, type **print_variable_group links_to_layout**.

SEE ALSO

lbo_cells_in_regions(3)
links_to_layout_variables(3)

mcmm_high_capacity_effort_level

Controls the behavior of Multi-corner/Multi_mode (MCMM) scenario reduction.

TYPE

Float. Valid range is between 0 and 10.

DEFAULT

0

GROUP

MCMM

DESCRIPTION

This variable controls how aggressively the MCMM scenario reduction feature (see **get_dominant_scenarios (2)** and **mcmm_enable_high_capacity_flow(3)**) reduces the number of dominant scenarios. In its default setting (0), any scenario with a violation that is worst across all scenarios is included in the dominant set of scenarios. As you increase the value of this variable, scenario reduction adjusts its criteria for comparing the slack of a violating object across different scenarios. This allows for further reduction of the dominant scenario set, but can implies that few violations will not be fixed during optimization.

If you set this variable to a value smaller than 0, scenario reduction will be performed using an effort level of 0. If you set this variable to a value larger than 10, scenario reduction will be performed using an effort level of 10.

SEE ALSO

set_active_scenarios(2)
all_scenarios(2)
all_active_scenarios(2)

monitor_cpu_memory

Displays the CPU time, elapsed time, and peak memory usage before and after each core command.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When **monitor_cpu_memory** is true, the CPU time, elapsed time, and peak memory usage are printed out before and after each major command, such as **place_opt** and **clock_opt**.

CPU time and elapsed time are the total time spent on the main process and all its child processes or multiple threads. For example, when you use the **-num_cpus** option, the reported CPU time includes the sum of the CPU usage of the main process and all child processes or threads, similar to the output of the **cputime -self -child** or **cputime -all** command.

Peak memory is defined as the memory high-water mark of the main process and its child processes.

To determine the current value of this variable, use **printvar monitor_cpu_memory**.

When you use the **-num_cpus** option, the tool runs multiple threads. Currently the operating system measures the CPU time by adding the usage for all threads. It does not take into account parallelization of the threads. This means that the report might show a larger CPU time than elapsed time when you specify the **-num_cpus** option.

The elapsed time (wall-clock time) depends on many external factors and can vary for every session. It depends on factors such as the I/O traffic, the network traffic, RAM and swap usage, and the other processes running on the same machine. When the elapsed time is much longer than the CPU time, it might point out an inefficiency of the computing environment, such as insufficient memory, too many processes running on the same machine, or a slow network. The only way to make the elapsed time close to the CPU time is to run only one session on a machine with enough RAM and using the local disk.

EXAMPLE

If set the **monitor_cpu_memory** variable to true, the tool outputs a PSYN-508 message in the log file after each major command. A sample PSYN-508 message is shown below:

```
Information:CPU: ### s (## hr) ELAPSE: ### s (## hr) MEM-PEAK: ### mb Mon Oct 28
```

```
12:20:01 2008 (PSYN-508)
```

SEE ALSO

```
mem(2)
cputime(2)
```

multibit_variables

SYNTAX

string **bus_multiple_separator_style** = ","

DESCRIPTION

These variables directly affect the multibit mapping and optimization capability in **compile** command.

For a list of these variables and their current values, type **print_variable_group multibit**. To view this manual page online, type **help multibit_variables**. To view an individual variable description, type **help *var***, where *var* is the variable name.

SEE ALSO

compile(2), **hdlin_infer_multibit**(3)

mux_auto_inferring_effort

Specifies the MUX inferring effort level.

TYPE

integer

DEFAULT

2

GROUP

fpga_variables

DESCRIPTION

The **mux_auto_inferring_effort** variable controls the MUX inferring effort of Design Compiler FPGA. Valid values are **0** through **6**. The default value is **2**. The larger the integer value, the more MUX is inferred.

mv_allow_ls_on_leaf_pin_boundary

Sets the variable to **true** to allow level-shifter insertion on leaf pin (such as macro cell pin) boundaries.

TYPE

Boolean

DEFAULT

false

GROUP

mv

DESCRIPTION

This variable controls whether or not to allow level-shifter insertion on leaf pin boundaries that are not power domain boundaries. When a macro cell is operating at a voltage different from its surrounding logic, and you want level shifters to be inserted at the interface, the recommended flow is to define a power domain around the macro cell by specifying the macro cell as the root cell of a power domain.

If you do not define the macro cell as the root cell, level shifters are not inserted at the interface, because the interface is not a power domain boundary. By default, level shifters are only inserted at power domain boundaries.

If you are unable to define a power domain around the macro cell, but still require level shifters to be inserted, you can use the variable to enable the non-default behavior.

SEE ALSO

compile_ultra(2)

mw_cell_name

Contains the Milkyway design cell name.

TYPE

string

DEFAULT

""

DESCRIPTION

Contains the Milkyway design cell name. The default is the empty string. If given, **read_mdb** reads in the cell specified by this variable, or **read_mdb** modifies this variable to indicate the cell, view, and version of the MDB cell read in. The **write_mdb** command searches for this variable when writing data to an MDB cell. If this variable is set, it writes to the cell specified by this variable, or it writes to a cell using the current design name.

SEE ALSO

mw_design_library

Contains the Milkyway design library.

TYPE

string

DEFAULT

""

DESCRIPTION

Contains the Milkyway design libraries. The default is the empty string. By setting this variable, the Milkyway design will be stored in the directory specified by this variable. If this variable is set, it will be used by the **read_milkyway**, **write_milkyway** and **create_mw_design** commands.

This variable needs to be set for a smooth data transfer to Milkyway.

SEE ALSO

write_milkyway(2)

mw_disable_escape_char

Specifies to disable the escape characters for hierarchy delimiter.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable controls the behavior of escape characters for hierarchies. This variable is used only by the **read_mdb** and **write_mdb** commands.

The default value is always to escape the hierarchy delimiters, if they are part of names.

If set to "true", the behavior is similar to setting "No Backslash Insertion to avoid Hier Name Collisions" in Astro.

Make sure that the design does not have name collisions.

SEE ALSO

mw_hdl_bus_dir_for_undef_cell

Specify how to determine the bus direction for undefined cell.

TYPE

Integer

DEFAULT

0

GROUP

hdl_variables

DESCRIPTION

If any port of any undefined cell is a bus, verilog2cel verilog reader creates a bus port for the undefined cell and instantiates it. The direction of the bus is specified via this variable.

For example, if an undefined cell is instantiated like the following:

```
sub si (.port(net[x:y]),...);
```

the number of bus bits = r = |x-y|+1

defines the bus origin according to the value given:

```
0 (From Connection)
x > y type [r:0] port
x < y type [0:r] port

1 (Descending) type [r:0] port

2 (Ascending) type [0:r] port
```

To determine the current value of this variable, use **printvar variable_name_filler**. For a list of all **hdl** variables and their current values, use the **print_variable_group hdl** command.

SEE ALSO

enable_cell_based_verilog_reader(3)
hdl_variables(3)

mw_hdl_expand_cell_with_no_instance

This variable determines whether to expand netlist cells without instances or not.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

This variable determines whether verilog2cel verilog reader to expands netlist cells that do not contain physical descriptions or not. Turn on this option if you have netlist instances containing only nets. If you do not turn on this option, and you have netlist instances with no physical description, expansion fails. The default is off.

To determine the current value of this variable, use **printvar mw_hdl_expand_cell_with_no_instance**. For a list of all **hdl** variables and their current values, use the **print_variable_group hdl** command.

SEE ALSO

hdl_variables(3)
enable_cell_based_verilog_reader(3)

mw_hdl_verilogel_variables

TCL Variables for the Milkyway cell based verilog reader (verilog2cel). Specify certain variables so that to make verilog2cel behaves as desired.

SYNTAX

```
mw_hdl_allow_dirty_netlist
mw_hdl_create_multi_pg_net
mw_hdl_bus_dir_for_undef_cell
mw_hdl_expand_cell_with_no_instance
mw_hdl_stop_at_FRAM_cells
```

Boolean **enable_cell_based_verilog_reader** = "false" (default)
Boolean **mw_hdl_allow_dirty_netlist** = "false" (default)
Boolean **mw_hdl_create_multi_pg_net** = "true" (default)
Boolean **mw_hdl_bus_dir_for_undef_cell** = 0 (default)
Boolean **mw_hdl_expand_cell_with_no_instance** = "false" (default)
Boolean **mw_hdl_stop_at_FRAM_cells** = "false" (default)

DESCRIPTION

These variables affect the **read_verilog** command when invoking Milkyway cell based verilog reader (verilog2cel) to import designs. Besides these variables, **mw_logic1_net** and **mw_logic0_net** and **mw_current_design** will also be used to indicate the net names for power(1'b1) and ground(1'b0).

enable_cell_based_verilog_reader
Enable Milkyway cell based verilog reader. See **enable_cell_based_verilog_writer(3)** for details.

mw_hdl_allow_dirty_netlist
Enable verilog2cel to handle dirty netlist. See **mw_hdl_allow_dirty_netlist(3)** for details.

mw_hdl_create_multi_pg_net
Generate multiple power and ground nets. See **mw_hdl_create_multi_pg_net(3)** for details.

mw_hdl_bus_dir_for_undef_cell
Specify the bus direction for undefined cell. See **mw_hdl_bus_dir_for_undef_cell(3)** for details.

mw_hdl_expand_cell_with_no_instance
This will enable verilog2cel to expand cell hierarchy which has no cell instance. See **mw_hdl_expand_cell_with_no_instance(3)** for details.

mw_hdl_stop_at_FRAM_cells
Tell verilog2cel to honor FRAM view cell. See **mw_hdl_stop_at_FRAM_cells(3)** for details.

SEE ALSO

mw_logic0_net (3), **mw_logic1_net** (3), **read_verilog** (2).

mw_hvo_core_filler_cells

Generate core filler cell instances for HVO.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Indicate Core Filler Cell instances to be generated in hierarchical verilog. The HVO writer will generate Core Filler Cell instances by default.

SEE ALSO

write(2)
mw_hvo_variables(3)

mw_hvo_corner_pad_cells

Generate corner pad cell instances for HVO.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Indicate Corner Pad Cell instances to be generated in hierarchical verilog. The HVO writer will generate Corner Pad Cell instances by default.

SEE ALSO

write(2)
mw_hvo_variables(3)

mw_hvo_diode_ports

Generate diode ports for HVO.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Indicate diode ports to be generated for cell instances in hierarchical verilog. The option is off by default to skip generating diode ports.

SEE ALSO

write(2)
mw_hvo_variables(3)

mw_hvo_dump_master_names

```
Specify the cell instance masters that must be dumped out for HVO.
```

TYPE

```
string
```

DEFAULT

```
""
```

DESCRIPTION

```
Specify the names of cell instance masters that must be dumped out regardless other
options.
```

SEE ALSO

```
write(2)
mw_hvo_variables(3)
```

mw_hvo_empty_cell_definition

Generate definition of empty cells for HVO.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Indicate to generate empty module definitions for the leaf level cells such as standard cells and macro cells. When this option is switched on, the command generates module definitions for all the ".FRAM view only" cells that contain port definitions but no internal cell instances. The HVO writer will skip empty module definitions by default.

SEE ALSO

write(2)
mw_hvo_variables(3)

mw_hvo_generate_macro_definition

Generate macro declarations for HVO.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Indicate module definitions to be generated for all the macros with hierarchy information in the CEL view. The HVO writer will skip module definitions by default.

SEE ALSO

write(2)
mw_hvo_variables(3)

mw_hvo_output_onezero_for_pg

Generate 1’b1 for power and 1’b0 for ground nets for HVO.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Indicate decimal value 1’b1 for power and 1’b0 for ground nets to be generate in hierarchical verilog. By default this switch is on. If turned off, the HVO writer will generate the power (use **mw_logic1_net** by default) and ground (use **mw_logic0_net** by default) net names.

SEE ALSO

write(2)
mw_hvo_variables(3)
mw_logic0_net(3)
mw_logic1_net(3)

mw_hvo_output_wire_declaration

Generate wire declarations of nets for HVO.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Indicate wire declaration to be prevented from being generated in hierarchical verilog. Only scalar wires are controlled by the option. For vector wires, a declaration with vector limits is generated regardless of this switch. The HVO writer will skip wire declaration by default.

SEE ALSO

write(2)
mw_hvo_variables(3)

mw_hvo_pad_filler_cells

Generate pad filler cell instances for HVO.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Indicate Pad Filler Cell instances to be generated in hierarchical verilog. The HVO writer will generate Pad Filler Cell instances by default.

SEE ALSO

write(2)
mw_hvo_variables(3)

mw_hvo_pg_nets

Generate power and ground nets for HVO.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Indicate all power and ground nets to be generated for cell instances, module instances and module definitions including the top-most module. For each power or ground net in the cell, there is a port generated for the module instances and module definitions. The HVO writer will generate power and ground nets by default.

SEE ALSO

write(2)
mw_hvo_variables(3)

mw_hvo_pg_ports

Generate power and ground ports for HVO.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Indicate all power and ground ports to be generated for cell instances. The HVO writer will skip power and ground ports by default.

SEE ALSO

write(2)
mw_hvo_variables(3)

mw_hvo_split_bus

Generate individual bus bits for HVO.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Indicate all the bus (vector) ports to be generated as individual bits. The port names are treated as scalar port names and are escaped. The HVO writer will not generate individual bits for bus ports by default.

SEE ALSO

write(2)
mw_hvo_variables(3)

mw_hvo_strip_backslash_before_hiersep

Do not generate backslashes before hierarchy seperators for HVO.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Indicate the backslash character preceding a hierarchy separator for all the instances to be removed. The option is selected by default.

SEE ALSO

write(2)
mw_hvo_variables(3)

mw_hvo_unconnected_cells

Generate unconnected cell instances for HVO.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Indicate unconnected cells with no ports connected except for power and ground ports to be generated in hierarchical verilog. The HVO writer will generate unconnected cells by default.

SEE ALSO

write(2)
mw_hvo_variables(3)

mw_hvo_unconnected_ports

Generate unconnected cell ports for HVO.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Indicate unconnected ports, including those ports on module instances, macro instances and standard cell instances, to be generated. If this options is true, all the unconnected ports are generated with a "SYNOPSYS_UNCONNECTED_%d" net where "%d" is the incrementally increased index number. The HVO writer will skip unconnected ports by default.

SEE ALSO

write(2)
mw_hvo_variables(3)

mw_hvo_variables

TCL Variables for the Milkyway cell based verilog (HVO) writer.

SYNTAX

Boolean **enable_cell_based_verilog_writer** = "false" (default)
Boolean **mw_hvo_pg_ports** = "false" (default)
Boolean **mw_hvo_pg_nets** = "true" (default)
Boolean **mw_hvo_split_bus** = "false" (default)
Boolean **mw_hvo_empty_cell_definition** = "false" (default)
Boolean **mw_hvo_corner_pad_cells** = "true" (default)
Boolean **mw_hvo_pad_filler_cells** = "true" (default)
Boolean **mw_hvo_core_filler_cells** = "true" (default)
Boolean **mw_hvo_unconnected_cells** = "true" (default)
Boolean **mw_hvo_unconnected_ports** = "false" (default)
Boolean **mw_hvo_strip_backslash_before_hiersep** = "true" (default)
Boolean **mw_hvo_diode_ports** = "false" (default)
Boolean **mw_hvo_output_wire_declaration** = "false" (default)
Boolean **mw_hvo_output_onezero_for_pg** = "true" (default)
Boolean **mw_hvo_generate_macro_definition** = "false" (default)
string **mw_hvo_dump_master_names** = "" (default)

DESCRIPTION

These variables affect the **write** command when invoking Milkyway cell based verilog writer (HVO) to export design. Besides these variables, **mw_logic1_net** and **mw_logic0_net** will also be used by HVO to indicate the net names for power(1'b1) and ground(1'b0).

enable_cell_based_verilog_writer
    Enable Milkyway cell based verilog writer. See **enable_cell_based_verilog_writer(3)** for details.

mw_hvo_pg_ports
    Generate power and ground ports. See **mw_hvo_pg_ports(3)** for details.

mw_hvo_pg_nets
    Generate power and ground nets. See **mw_hvo_pg_nets(3)** for details.

mw_hvo_split_bus
    Generate individual bus bits. See **mw_hvo_split_bus(3)** for details.

mw_hvo_empty_cell_definition
    Generate definition of empty cells. See **mw_hvo_empty_cell_definition(3)** for details.

mw_hvo_corner_pad_cells
    Generate corner pad cell instances. See **mw_hvo_corner_pad_cells(3)** for details.

mw_hvo_pad_filler_cells
        Generate pad filler cell instances. See **mw_hvo_pad_filler_cells(3)** for details.

mw_hvo_core_filler_cells
        Generate core filler cell instances. See **mw_hvo_core_filler_cells(3)** for details.

mw_hvo_unconnected_cells
        Generate unconnected cell instances. See **mw_hvo_unconnected_cells(3)** for details.

mw_hvo_unconnected_ports
        Generate unconnected cell ports. See **mw_hvo_unconnected_ports(3)** for details.

mw_hvo_strip_backslash_before_hiersep
        Do not generate backslashes before hierarchy seperators. See **mw_hvo_strip_backslash_before_hiersep(3)** for details.

mw_hvo_diode_ports
        Generate diode ports. See **mw_hvo_diode_ports(3)** for details.

mw_hvo_output_wire_declaration
        Generate wire declarations of nets. See **mw_hvo_output_wire_declaration(3)** for details.

mw_hvo_output_onezero_for_pg
        Generate 1’b1 for power and 1’b0 for ground nets. See **mw_hvo_output_onezero_for_pg(3)** for details.

mw_hvo_generate_macro_definition
        Generate macro declarations. See **mw_hvo_generate_macro_definition(3)** for details.

mw_hvo_dump_master_names
        Specify the cell instance masters that must be dumped out. See **mw_hvo_dump_master_names(3)** for details.

SEE ALSO

mw_logic0_net (3), **mw_logic1_net** (3), **write** (2).

mw_logic0_net

Contains the equivalent logic0 net for the design.

TYPE

string

DEFAULT

VSS

DESCRIPTION

Contains the milkyway logic0(1’b0) net for the design. The default is "VSS".

mw_logic1_net

Contains the equivalent logic1 net for the design.

TYPE

string

DEFAULT

VDD

DESCRIPTION

Contains the milkyway logic1(1’b1) net for the design. The default is "VDD".

mw_reference_library

Contains the Milkyway reference libraries.

TYPE

list

DEFAULT

""

DESCRIPTION

Contains the Milkyway reference libraries. The default is the empty string. By setting this variable, the search_path and the physical_library will be enhanced to use the Milkyway libraries.

SEE ALSO

net_attributes

Contains attributes that can be placed on a net.

DESCRIPTION

Contains attributes that can be placed on a net.

To set an attribute, use the command identified in the individual description of that attribute. If an attribute is "read-only," the user cannot set it.

To determine the value of an attribute, use the **get_attribute** command. To remove attributes, use the **remove_attribute** command.

For a more detailed explanation of an attribute, refer to the manual pages of the appropriate **set** command. For information on all attributes, refer to the **attributes** manual page.

Net Attributes

ba_net_resistance
: A floating point number that specifies the back-annotated net resistance on a net. Set with **set_resistance**.

dont_touch
: Identifies nets to be excluded from optimization. Values are *true* (the default) or *false*. Nets with the **dont_touch** attribute set to *true* are not modified or replaced during **compile**. Set with **set_dont_touch**.

is_test_circuitry
: Set by **insert_dft** on the scan cells and nets added to a design during the addition of test circuitry. This attribute is read-only and cannot be set by the user.

load *
: A floating point number that specifies the wire load value on a net. The total load on a net is the sum of all the loads on pins, ports, and wires associated with that net. This attribute represents only the wire load; pin and port loads are not included in the attribute value unless the attribute *subtract_pin_load* is set to **true** for the same net. Set with **set_load**.

static_probability
: A floating point number that specifies the percentage of time that the signal is in the logic 1 state; this information is used by **report_power**. If this attribute is not set, **report_power** will use the default value of 0.5, indicating that the signal is in the logic 1 state half the time. Set with **set_switching_activity**.

subtract_pin_load
: Causes **compile** to reduce the wire load value of a net by an amount equal to its pin load. Specifies that the **load** attribute includes the capacitances of all pins on the net. If the resulting wire load is negative, it is set to zero. Set with **set_load -subtract_pin_load**.

toggle_rate

A positive floating point number that specifies the toggle rate; that is, the number of zero-to-one and one-to-zero transitions within a library time unit period. This information is used by **report_power**; if this attribute is not set, **report_power** will use the default value of 2*(static_probability)(1 - static_probability). The default will be scaled by any associated clock signal (if one is available). Set with **set_switching_activity**.

wired_and

One of a set of two wired logic attributes that includes **wired_or**. When present and set to *true*, **wired_and** determines that the associated net has more than one driver and implements a wired AND function. Wired logic attributes cannot be manually set by the user. To cause wired logic attributes to be added to a netlist design that contains multiply-driven nets, you have two alternatives: 1. execute **compile** or **translate** on the design; or 2. specify the wired logic types using a resolution function in the HDL file.

wired_or

One of a set of two wired logic attributes that includes **wired_and**. When present and set to *true*, **wired_or** determines that the associated net has more than one driver and implements a wired OR function. Wired logic attributes cannot be manually set by the user. To cause wired logic attributes to be added to a netlist design that contains multiply-driven nets, you have two alternatives: 1. execute **compile** or **translate** on the design; or 2. specify the wired logic types using a resolution function in the HDL file.

SEE ALSO

get_attribute(2)
remove_attribute(2)
attributes(3)

optimize_reg_always_insert_sequential

Controls whether the **optimize_registers** command will remove and reinsert the sequential elements in the circuits even if no register was moved.

TYPE

Boolean

DEFAULT

false

GROUP

retiming_variables

DESCRIPTION

When set to ’true’, **optimize_registers** will remove all existing movable elements from the design and reinsert and remap them, even if no registers were moved to improve delay or area cost. By default (value ’false’) **optimize_registers** leaves sequential elements unchanged, if none of them are moved to improve delay or area cost. If your design contains register trees which are not build optimally setting this variable to ’true’ before **optimize_registers** is executed might reduce the register count.

SEE ALSO

optimize_registers(2)

optimize_reg_max_time_borrow

Specifies the maximum amount of time borrowing at all latches when retiming latches using the **optimize_registers** command. A negative value means there is no limit on borrowing other than the one resulting from the clock period.

TYPE

float

DEFAULT

-1048576.0

GROUP

retiming_variables

DESCRIPTION

This variable limits the amount of borrowing that can occur at latches while performing retiming optimization. By default, during retiming, you can borrow latches up to one half of the clock period at a latch in order to avoid having negative slack. A negative value means there is no limit on the amount (other than one half of the clock period) of borrowing allowed.

SEE ALSO

optimize_registers(2)

optimize_reg_retime_clock_gating_latches

Specifies whether to move clock gating latches during retiming of latches.

TYPE

Boolean

DEFAULT

false

GROUP

retiming_variables

DESCRIPTION

The **optimize_registers** command allows to retime latches when using the -latch option, but will by default not move clock-gating latches. Clock-gating latches are latches that drive clock gating cells and are used to prevent glitches on gated clocks. If for some reason you want to move clock-gating latches during retiming you can set this variable to true. If you want to control individual clock gating latches you can set the variable to true and use the set_dont_touch or set_transform_for_retiming with value ’dont_retime’.

SEE ALSO

optimize_registers(2)

pdefout_diff_original

Used in conjunction with the option -*new_cells_only* of the command **write_clusters**.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Used in conjunction with the option -*new_cells_only* of the command **write_clusters**.

When this variable is set to *true*, the command **write_clusters -new_cells_only** writes the new cells that are not in the original pdef file. It determines the new cells by comparing the cell names.

When this variable is set to *false*, the command **write_clusters -new_cells_only** writes the cells that were added during the last run of the **reoptimize_design** command.

The default value of this variable is *true*.

SEE ALSO

physical_bus_attributes

Contains attributes related to physical bus.

DESCRIPTION

Contains attributes related to physical bus.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified physical bus. Specified with **list_attribute -class physical_bus -application**, the definition of attributes can be listed.

Physical Bus Attributes

cell_id

Specifies Milkyway design ID in which a physical bus object is located.
This attribute is read-only.

name

Specifies name of a physical bus object.
This attribute is read-only.

num_nets

Specifies count of nets in a physical bus object.
The nets can be collected by **get_nets -of_object <physical_bus_object>**.
This attribute is read-only.

object_class

Specifies object class name of a physical bus object, which is **physical_bus**.
This attribute is read-only.

object_id

Specifies object ID in Milkyway design file.
This attribute is read-only.

SEE ALSO

get_attribute(2),
list_attribute(2),
report_attribute(2).

physopt_area_critical_range

Specifies a margin of slack for cells during area optimization. If a cell has a slack less than the area critical range, area optimization is not done for the cell.

TYPE

float

DEFAULT

-1.04858e+06

GROUP

physopt

DESCRIPTION

The area critical range specifies a margin of slack for cells during area optimization. If a cell has a slack less than the area critical range, area optimization is not done for the cell. The default value is minus infinity; that is, all cells are optimized for area during area recovery.

Use the following command to determine the current value of the variable:

```
   prompt> printvar physopt_area_critical_range
```

SEE ALSO

physopt_cpu_limit

This variable is obsolete and cannot be enabled. Please use set_physopt_cpulimit_options command instead.

TYPE

float

DEFAULT

0

GROUP

physopt

DESCRIPTION

Please remove the obsolete variable, and replace it with the new command, **set_physopt_cpulimit_options**. For details, please check the **set_physopt_cpulimit_options** man page.

SEE ALSO

physopt_create_missing_physical_libcells

Directs Physical Compiler to create dummy physical descriptions of missing physical library cells.

TYPE

Boolean

DEFAULT

false

GROUP

physopt

DESCRIPTION

This variable is used to enable Physical Compiler to create a dummy physical cell if a cell missing from the user-supplied physical libraries.

Physical Compiler uses a simple physical description for these dummy physical cells. This is mainly required in the exploration flow so that Physical Compiler will not stop when you do not have the final physical libraries. This is not intended as a replacement for specifying the correct physical libraries. For a final and accurate solution, you must specify the correct physical libraries.

The dummy physical cells created by Physical Compiler are not persistent. They exist in the current session only and are not stored in the .db database. These cells will be re-created in a new Physical Compiler session, if they are still required.

The **physopt_create_missing_physical_libcells** variable is valid only in Physical Compiler.

To determine the current value of this variable, enter the following command:

```
psyn_shell-t> printvar physopt_create_missing_physical_libcells
```

SEE ALSO

physopt_enable_extractor_rc

Enables and disables the support of extractor-based RC computation.

TYPE

Boolean

DEFAULT

true

GROUP

physopt

DESCRIPTION

The **physopt_enable_extractor_rc** variable enables the support of extractor-based RC computation. The default value is **true**.

If you want to disable the support of extractor-based RC computation in your design, set the value of this variable as **false**.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar physopt_enable_extractor_rc
```

SEE ALSO

physopt_enable_router_process

Enables and disables the use of a separate process to perform routing.

TYPE

Boolean

DEFAULT

true

GROUP

physopt

GROUP

physopt

DESCRIPTION

This variable enables the use of a separate process to perform routing when set to true and disables it when set to false.

SEE ALSO

physopt_enable_tlu_plus

Enables and disables the support of TLU+ based RC computation.

TYPE

Boolean

DEFAULT

true

GROUP

physopt

DESCRIPTION

The **physopt_enable_tlu_plus** variable enables the support of TLU+ based RC computation in your design. The default value is **true**.

If you want to disable the support of TLU+ based RC computation in your design, set the value of this variable as **false**.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar physopt_enable_tlu_plus
```

SEE ALSO

physopt_enable_tlu_plus_process

Enables and disables using a separate process to perform TLU+ based RC extraction.

TYPE

Boolean

DEFAULT

true

GROUP

physopt

DESCRIPTION

When you set the value of the variable to *true*, TLU+ based RC extraction is done in a separate process. This variable is only valid with post-route extraction with TLU+.

From 2007.03, the default setting is to run TLU+ based rc extraction in separate process automatically, it isn’t necessory to set this variable.

SEE ALSO

physopt_enable_via_res_support

Enables and disables the support of via resistance for virtual route RC estimation.

TYPE

Boolean

DEFAULT

false

GROUP

physopt

DESCRIPTION

Setting the value of this variable as **true** enables the support of via resistance for virtual route RC estimation. The default value is **false**.

If you want to disable the support of via resistance for virtual route RC estimation, set the value of this variable as **false** (the default value).

Use the following command to see the current value of this variable:

```
   prompt> printvar physopt_enable_via_res_support
```

physopt_hard_keepout_distance

Specifies the keepout distance used by the **physopt**, **create_placement**, and **legalize_placement** commands.

TYPE

float

DEFAULT

0

GROUP

physopt

DESCRIPTION

Use this variable to specify the keep-out distance used by the **physopt**, **create_placement**, and **legalize_placement** commands. Specify the distance in micron units.

To prevent congestion, it is useful to mark as a "hard keep-out area" an area in the design that surrounds a fixed macro, a capability this variable provides. If you specify an area as a hard keep-out area, Physical Compiler does not place any cells there. Define the area to be marked by setting a value for this variable, which the tool then uses to inflate the fixed cell's rectangle the same distance in all four directions.

Setting this variable is optional. The default value is 0.0.

To determine the variable's current value, type the following:

```
psyn_shell-t> printvar physopt_hard_keepout_distance
```

SEE ALSO

physopt_ignore_lpin_fanout

The tool ignore max fanout constraints which are specfied in the library.

TYPE

Boolean

DEFAULT

false

GROUP

physopt

DESCRIPTION

When this variable is turned on, tools will ignore max_fanout violatios from the constraints in the library. Only optimization ignores those constraints and report_constraint will still report the violation.

physopt_power_critical_range

Specifies a margin of slack for cells during leakage power optimization. If a cell has a slack less than the power critical range, power optimization will not be done for the cell.

TYPE

float

DEFAULT

-1.04858e+06

GROUP

physopt

DESCRIPTION

The power critical range specifies a margin of slack for cells during leakage power optimization. If a cell has a slack less than the power critical range, leakage power optimization will not be done for the cell. The default value is minus infinity; that is, all cells will be optimized for leakage power during the leakage power optimization phase of the **physopt** command.

SEE ALSO

set_max_leakage_power(2)

pin_attributes

Contains attributes placed on pins.

DESCRIPTION

Contains attributes that can be placed on a pin.

There are a number of commands used to set attributes; however, most attributes can be set with the **set_attribute** command. If the attribute definition specifies a **set** command, use it to set the attribute. Otherwise, use **set_attribute**. If an attribute is *read only*, it cannot be set by the user.

To determine the value of an attribute, use the **get_attribute** command. To remove attributes, use the **remove_attribute** command.

For a more detailed explanation of an attribute, refer to the manual pages of the appropriate **set** command. For information on all attributes, refer to the **attributes** manual page.

Pin Attributes

disable_timing
: Disables timing arcs. This has the same affect on timing as not having the arc in the library. Set with **set_disable_timing**.

fall_delay
: Specifies an offset from the falling edge of the ideal clock waveform. Set with **set_clock_skew -fall_delay**.

max_fall_delay
: A floating point value that specifies the maximum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

max_rise_delay
: A floating point value that specifies the maximum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

min_fall_delay
: A floating point value that specifies the minimum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

min_rise_delay
: A floating point value that specifies the minimum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

minus_uncertainty
: Specifies a negative uncertainty from the edges of the ideal clock waveform. Set with **set_clock_skew -minus_uncertainty**.

observe_pin
: Specifies the (internal) observe pin name of an LSI Logic scan macrocell (LSI CTV only). This attribute is used by the **write_test** command. Set with **set_attribute**.

pin_direction
: Specifies the direction of a pin. Allowed values are *in*, *out*, *inout*, or *unknown*. This attribute is **read only** and cannot be set by the user.

pin_properties
: Lists valid EDIF property values to be attached to different versions of the output pin. The EDIF property values correspond to different output emitter-follower resistance values on the output pin. For details about the use of this attribute, refer to the *Library Compiler Reference Manual*. Set with **set_attribute**.

plus_uncertainty
: Specifies a positive uncertainty from the edges of the ideal clock waveform. Affects all sequential cells in the transitive fanout of this port. Set with **set_clock_skew -plus_uncertainty**.

propagated_clock
: Specifies that the clock edge times be delayed by propagating the values through the clock network. If this attribute is not present, ideal clocking is assumed. Set with **set_clock_skew -propagated**.

rise_delay
: Specifies an offset from the rising edge of the ideal clock waveform. Set with **set_clock_skew -rise_delay**.

set_pin
: Specifies the (internal) set pin name of an LSI Logic scan macrocell (LSI CTV only). This attribute is used by the **write_test** command. Set with **set_attribute**.

signal_type
: Used to indicate that a pin or port is of a special type, such as a **clocked_on_also** port in a master/slave clocking scheme, or a **test_scan_in** pin for scan-test circuitry. Set with **set_signal_type**.

static_probability
: A floating point number that specifies the percentage of time that the signal is in the logic 1 state; this information is used by **report_power**. If this attribute is not set, **report_power** will use the default value of 0.5, indicating that the signal is in the logic 1 state half the time. Set with **set_switching_activity**.

test_assume
: A string that represents a constant logic value to be assumed for specified pins throughout test design rule checking, test pattern generation, and fault simulation by **check_test**, **create_test_patterns**, and **fault_simulate**. "1", "one", or "ONE" specifies a constant value of logic one; "0", "zero", or "ZERO" specifies a constant value of logic zero. Use **report_test -assertions** for a report on objects that have the **test_assume** attribute set. Set with **set_test_assume**.

test_dont_fault
: Specifies pins not faulted during test pattern generation. If no command options are specified, this attribute is set for both stuck-at-0 and stuck-at-1 faults. Set with **set_test_dont_fault**.

test_initial
: A string that represents an initial logic value to be assumed for specified pins at the start of test design rule checking and fault simulation by **check_test** and **fault_simulate**. "1", "one", or "ONE" specifies an initial value of logic one; "0", "zero", or "ZERO" specifies an initial value of logic zero. Use **report_test -assertions** for a report on objects that have the **test_initial** attribute set. Set with **set_test_initial**.

test_isolate
: Indicates that the specified sequential cells, pins, or ports are to be logically isolated and considered untestable during test design rule checking by **check_test**. When this attribute is set on a cell, it is also placed on all pins of that cell. Do not set this attribute on a hierarchical cell. Use **report_test -assertions** for a report on isolated objects. Set with **set_test_isolate**.
Note: Setting this attribute suppresses the warning messages associated with the isolated objects.

test_require
: Specifies a constant, fixed logic value that a pin is required to have during scan test vector generation. The pin maintains the same value for each test vector generated. Use **report_test -assertions** for a report on objects that have the **test_require** attribute set. Set with **set_test_require**.

test_routing_position
: Specifies the preferred routing order of the scan-test signals of the identified cells. Set with **set_test_routing_order**.

toggle_rate
: A positive floating point number that specifies the toggle rate; that is, the number of zero-to-one and one-to-zero transitions within a library time unit period. This information is used by **report_power**; if this attribute is not set, **report_power** will use the default value of 2*(static_probability)(1 - static_probability). The default will be scaled by any associated clock signal (if one is available). Set with **set_switching_activity**.

true_delay_case_analysis
: Specifies a value to set all or part of an input vector for **report_timing -true** and **report_timing -justify**. Allowed values are *0*, *1*, *r* (rise, X to 1), and *f* (fall, X to 0). Set with **set_true_delay_case_analysis**.

SEE ALSO

get_attribute(2)
remove_attribute(2)
set_attribute(2)
attributes(3)

pin_shape_attributes

Contains attributes related to pin shape.

DESCRIPTION

Contains attributes related to pin shape.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified physical bus. Specified with **list_attribute -class pin_shape -application**, the definition of attributes can be listed.

Pin Shape Attributes

access_direction

Specifies the accessible direction(s) of a pin shape.
This attribute is read-only.
The valid values can be:

- **Left** indicates the object can be accessed from left.
- **Right** indicates the object can be accessed from right.
- **Up** indicates the object can be accessed from above.
- **Down** indicates the object can be accessed from below.
- **Left Right** indicates the object can be accessed from left and right.
- **Left Up** indicates the object can be accessed from left and above.
- **Left Down** indicates the object can be accessed from left and below.
- **Right Up** indicates the object can be accessed from right and above.
- **Right Down** indicates the object can be accessed from right and below.
- **Up Down** indicates the object can be accessed from above and below.
- **Left Right Up** indicates the object can not be access from below.

- **Left Right Down** indicates the object can not be access from above.

- **Left Up Down** indicates the object can not be access from right.

- **Right Up Down** indicates the object can not be access from left.

- **Left Right Up Down** indicates the object can be accessed from all directions.

base_name

Specifies the base name of a pin shape. The **base_name** of a pin shape is similar with the **name** of the corresponding terminal in the child MW design. This attribute is read-only.

bbox

Specifies the bounding-box of a pin shape. The **bbox** is represented by a **rectangle**.
The format of a *rectangle* specification is {{llx lly} {urx ury}}, which specifies the lower-left and upper-right corners of the rectangle.
The *bbox* of a pin shape is calculated by the **origin** and **orientation** of its cell and the actual **bbox** of its corresponding terminal from the child MW design.
This attribute is read-only.

bbox_ll

Specifies the lower-left corner of the bounding-box of a pin shape.
The **bbox_ll** is represented by a **point**. The format of a *point* specification is {x y}.
You can get the **bbox_ll** of a pin shape by accessing the first element of its **bbox**.
This attribute is read-only.

bbox_llx

Specifies x coordinate of the lower-left corner of the bounding-box of a pin shape.
The data type of **bbox_llx** is double.
This attribute is read-only.

bbox_lly

Specifies y coordinate of the lower-left corner of the bounding-box of a pin shape.
The data type of **bbox_lly** is double.
This attribute is read-only.

bbox_ur

Specifies the upper-right corner of the bounding-box of a pin shape.
The **bbox_ur** is represented by a **point**. The format of a *point* specification is {x y}.
You can get the **bbox_ur** of a pin shape by accessing the second element of its

bbox.
This attribute is read-only.

bbox_urx

Specifies x coordinate of the upper-right corner of the bounding-box of a pin shape.
The data type of **bbox_urx** is double.
This attribute is read-only.

bbox_ury

Specifies y coordinate of the upper-right corner of the bounding-box of a pin shape.
The data type of **bbox_ury** is double.
This attribute is read-only.

cell_id

Specifies Milkyway design ID in which a pin shape object is located.
This attribute is read-only.

direction

Specifies the direction of a pin shape.
The valid values can be: in, out, inout and tristate.
This attribute is read-only.

full_name

Specifies the full name of a pin shape.
The full name of a pin shape consists of the full name of its owned cell and the name of the corresponding terminal from child MW design.
This attribute is read-only.

is_fixed

Indicates whether a pin shape is fixed or not.
This attribute is read-only.

layer

Specifies the layer name of a pin shape.
This attribute is read-only.

name

Specifies name of a pin shape. It's identical as the attribute **full_name**.
This attribute is read-only.

number_of_points

Specifies the number of points to illustrate the boundary of a pin shape.
You can refer to the attribute **points**. The list length of **points** is the value of **number_of_points**.
This attribute is read-only.

object_class

Specifies object class name of a pin shape, which is **pin_shape**.
This attribute is read-only.

points

Specifies points of the boundary of a pin shape. A pin shape can be a rectangle, a rectilinear polygon, or multiple rectangles.

When a pin shape is either a rectangle or a rectilinear polygon, its **points** is represented by a list of points. The last element of the list is the same as the first element.
When a pin shape consists of multiple rectangles, its **points** is represented by a list of points of rectangles. Every five points represent one rectangle.
The *points* of a pin shape is calculated by the **origin** and **orientation** of its cell and the actual **points** of its corresponding terminal from the child MW design.
This attribute is read-only.

status

Specifies the PR status of a pin shape.
The valid values are: fixed, placed, cover, and unplaced.
This attribute is read-only.

SEE ALSO

get_attribute(2),
list_attribute(2),
report_attribute(2).

placement_blockage_attributes

Contains attributes related to placement blockage.

DESCRIPTION

Contains attributes related to placement blockage.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified object. Specified with **list_attribute -class placement_blockage -application**, the definition of attributes can be listed.

Placement Blockage Attributes

affects

Specifies the affects of a placement blockage, which is placement.
The data type of **affects** is string.
This attribute is read-only.

area

Specifies area of a placement blockage.
The data type of **area** is float.
This attribute is read-only.

bbox

Specifies the bounding-box of a placement blockage. The **bbox** is represented by a **rectangle**.
The format of a *rectangle* specification is {{llx lly} {urx ury}}, which specifies the lower-left and upper-right corners of the rectangle.
The data type of **bbox** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

blocked_percentage

Specifies the percentage blockage for a partial blockage.
The data type of **blocked_percentage** is integer.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

cell_id

Specifies Milkyway design ID in which a placement blockage object is located.
The data type of **cell_id** is integer.
This attribute is read-only.

layer

Specifies layer name of a placement blockage.
The data type of **layer** is string.
Its valid values are:

- **PlaceBlockage**

- **SoftPlaceBlk**
- **pinBlockage**
- **MacroBlockage**
- **PartialPlaceBlk**

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

name
: Specifies name of a placement blockage object.
The data type of **name** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

object_class
: Specifies object class name of a placement blockage, which is **placement_blockage**.
The data type of **object_class** is string.
This attribute is read-only.

object_id
: Specifies object ID in Milkyway design file.
The data type of **object_id** is integer.
This attribute is read-only.

pin_blockage_layers
: Specifies the layers for which routing to pins are blocked. This attribute only applies on pin blockage type.
The data type of **pin_blockage_layers** is string.
This attribute is read-only.

type
: Specifies type of a placement blockage.
The data type of **type** is string.
Its valid values are:

- **hard**
- **soft**
- **pin**

- **hard_macro**

- **partial**

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

SEE ALSO

get_attribute(2),
list_attribute(2),
report_attribute(2),
set_attribute(2).

placer_enable_enhanced_router

Enables a mode of coarse placement in which congestion removal is done with the global router.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Enables a mode of coarse placement in which congestion removal is driven by the global router.

When this variable is set to false, the congestion removal in coarse placement is driven by an internal global route estimator.

For most designs, QoR does not change much when turning on this feature. But for some designs, you may see a significant congestion reduction when this feature is turned on.

An important change when using the variable is that the **set_congestion_options -layer** command is not supported when **placer_enable_enhanced_router** is enabled.

For example, to enable this feature, enter

```
prompt> set placer_enable_enhanced_router true
```

SEE ALSO

placer_max_cell_density_threshold

Enables a mode of coarse placement in which cells can clump together.

TYPE

float

DEFAULT

-1

DESCRIPTION

Enables a mode of coarse placement in which cells are not distributed evenly across the surface of the chip, but are allowed to clump together. The value you specify sets the threshold of how tightly the cells are allowed to clump. The value of 1.0 allows no gaps between cells.

A reasonable value is one that is above the background utilization of your design but below 1.0. For example, if your background utilization is 40%, or 0.4, a reasonable value for this variable is a value between 0.4 and 1.0. The higher the value, the more tightly the cells clump together.

For example, to set the threshold to 0.7, enter

```
   prompt> set placer_max_cell_density_threshold 0.7
```

SEE ALSO

placer_run_in_separate_process

Enables and disables the use of a separate process to perform placement.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable enables the use of a separate process to perform placement when set to true and disables it when set to false.

SEE ALSO

placer_soft_keepout_channel_width

Specifies a soft keepout distance that is used by the **place_opt** and **create_placement** commands.

TYPE

float

DEFAULT

0

DESCRIPTION

This variable is used to specify a soft keepout distance that is used by the **place_opt** and **create_placement** commands. The distance must be specified in micron units. If set, this variable causes the tool to automatically identify thin channels (regions) around fixed cells and blockages as soft keepout areas.

Note that putting too many cells in such a thin keepout area might potentially lead to poor results. If an area is identified as a soft keepout area, the tool tries not to put too many cells in that area.

If set, this variable causes the tool to mark various areas near fixed RAMs and blockages as soft keepout areas. The soft keepout areas are not generated for every fixed RAM or blockage. Such areas are generated for an object (fixed RAM or blockage) only if the distance between the object and the core area boundary is less than the soft keepout distance or if the distance between two such objects is less than the soft keepout distance (forming a thin channel between the objects).

Setting this variable is optional. If unspecified, the default is 0.0.

To determine the current value of this variable use the following command:

```
   prompt> printvar placer_soft_keepout_channel_width
```

SEE ALSO

plan_group_attributes

Contains attributes related to plan group.

DESCRIPTION

Contains attributes related to plan group.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified object. Specified with **list_attribute -class plan_group -application**, the definition of attributes can be listed.

Plan Group Attributes

aspect_ratio

Specifies the **height:width** ratio of a plan group.
The data type of **aspect_ratio** is double.
This attribute is read-only.

bbox

Specifies the bounding-box of a plan group. The **bbox** is represented by a **rectangle**.
The format of a *rectangle* specification is {{llx lly} {urx ury}}, which specifies the lower-left and upper-right corners of the rectangle.
The data type of **bbox** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_ll

Specifies the lower-left corner of the bounding-box of a plan group.
The **bbox_ll** is represented by a **point**. The format of a *point* specification is {x y}.
You can get the **bbox_ll** of a plan group, by accessing the first element of its **bbox**.
The data type of **bbox_ll** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_llx

Specifies x coordinate of the lower-left corner of the bounding-box of a plan group.
The data type of **bbox_llx** is double.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_lly

Specifies y coordinate of the lower-left corner of the bounding-box of a plan group.
The data type of **bbox_lly** is double.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_ur

Specifies the upper-right corner of the bounding-box of a plan group.
The **bbox_ur** is represented by a **point**. The format of a *point* specification is {x y}.
You can get the **bbox_ur** of a plan group, by accessing the second element of its **bbox**.
The data type of **bbox_ur** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_urx

Specifies x coordinate of the upper-right corner of the bounding-box of a plan group.
The data type of **bbox_urx** is double.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_ury

Specifies y coordinate of the upper-right corner of the bounding-box of a plan group.
The data type of **bbox_ury** is double.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bottom_padding

Specifies bottom-side width of interior paddings.
The data type of **bottom_padding** is double.
This attribute is read-only. You can use **create_fp_plan_group_padding** to set its value.

bottom_padding_external

Specifies bottom-side width of exterior paddings.
The data type of **bottom_padding_external** is double.
This attribute is read-only. You can use **create_fp_plan_group_padding** to set its value.

bottom_shielding

Specifies bottom-side widths of signal shielding inside a plan group.
The data type of **bottom_shielding** is string.
This attribute is read-only. You can use **create_fp_block_shielding** to set its value.

bottom_shielding_external

Specifies bottom-side widths of signal shielding outside a plan group.
The data type of **bottom_shielding_external** is string.
This attribute is read-only. You can use **create_fp_block_shielding** to set its value.

color

Specifies color to draw a plan group and its associated instances.
The data type of **color** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

is_fixed

Specifies that a plan group is in a fixed location, and the shaping will ignore it.
The data type of **is_fixed** is boolean.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

left_padding

Specifies left-side width of interior paddings.
The data type of **left_padding** is double.
This attribute is read-only. You can use **create_fp_plan_group_padding** to set its value.

left_padding_external

Specifies left-side width of exterior paddings.
The data type of **left_padding_external** is double.
This attribute is read-only. You can use **create_fp_plan_group_padding** to set its value.

left_shielding

Specifies left-side widths of signal shielding inside a plan group.
The data type of **left_shielding** is string.
This attribute is read-only. You can use **create_fp_block_shielding** to set its value.

left_shielding_external

Specifies left-side widths of signal shielding outside a plan group.
The data type of **left_shielding_external** is string.
This attribute is read-only. You can use **create_fp_block_shielding** to set its value.

logic_cell

Specifies name of the hierarchical cell associated with a plan group.
The data type of **logic_cell** is string.
This attribute is read-only.

name

Specifies name of a plan group object.
The data type of **name** is string.
This attribute is read-only.

number_of_hard_macro

Specifies number of hard macro cells inside a plan group.
The data type of **number_of_hard_macro** is integer.
This attribute is read-only.

number_of_pin

Specifies number of pins on the hierarchical cell associated with a plan group.
The data type of **number_of_pin** is integer.
This attribute is read-only.

number_of_standard_cell

Specifies number of standard cells inside a plan group.
The data type of **number_of_standard_cell** is integer.

This attribute is read-only.

object_class

Specifies object class name of a plan group, which is **plan_group**.
The data type of **object_class** is string.
This attribute is read-only.

points

Specifies point list of a plan group’s boundary.
The data type of **points** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

right_padding

Specifies right-side width of interior paddings.
The data type of **right_padding** is double.
This attribute is read-only. You can use **create_fp_plan_group_padding** to set its value.

right_padding_external

Specifies right-side width of exterior paddings.
The data type of **right_padding_external** is double.
This attribute is read-only. You can use **create_fp_plan_group_padding** to set its value.

right_shielding

Specifies right-side widths of signal shielding inside a plan group.
The data type of **right_shielding** is string.
This attribute is read-only. You can use **create_fp_block_shielding** to set its value.

right_shielding_external

Specifies right-side widths of signal shielding outside a plan group.
The data type of **right_shielding_external** is string.
This attribute is read-only. You can use **create_fp_block_shielding** to set its value.

target_utilization

Specified target utilization set on a plan group.
The data type of **target_utilization** is double.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

top_padding

Specifies top-side width of interior paddings.
The data type of **top_padding** is double.
This attribute is read-only. You can use **create_fp_plan_group_padding** to set its value.

top_padding_external

Specifies top-side width of exterior paddings.
The data type of **top_padding_external** is double.
This attribute is read-only. You can use **create_fp_plan_group_padding** to set its value.

top_shielding

Specifies top-side widths of signal shielding inside a plan group.
The data type of **top_shielding** is string.
This attribute is read-only. You can use **create_fp_block_shielding** to set its value.

top_shielding_external

Specifies top-side widths of signal shielding outside a plan group.
The data type of **top_shielding_external** is string.
This attribute is read-only. You can use **create_fp_block_shielding** to set its value.

utilization

Specifies the ratio of total area size of associated instances to the area of a plan group.
The data type of **utilization** is double.
This attribute is read-only.

SEE ALSO

create_fp_block_shielding(2),
create_fp_plan_group_padding(2),
get_attribute(2),
list_attribute(2),
report_attribute(2),
set_attribute(2).

plot_box

Causes a box to be drawn around the plot. The default is *false*.

TYPE

Boolean

DEFAULT

false

GROUP

plot_variables

DESCRIPTION

Causes a box to be drawn around the plot. The default is *false*.

To determine the current value of this variable, use **printvar plot_box**. For a list of all **plot** variables and their current values, use the **print_variable_group plot** command.

SEE ALSO

plot_variables(3)

plot_command

Specifies the operating system command that produces a hard copy of the plot.

TYPE

string

DEFAULT

lpr -Plw

GROUP

plot_variables

DESCRIPTION

Specifies the operating system command that produces a hard copy of the plot. The PostScript output of the **plot** command is piped as standard input (stdin) to the given command.

If this variable is not a valid system command, an operating system error occurs during **plot**. Default is "lpr -Plw -h". This variable is usually set only once for a site, in the system **.synopsys_dc.setup** initialization file.

To determine the current value of this variable, use **printvar plot_command**. For a list of all **plot** variables and their current values, use the **print_variable_group plot** command.

SEE ALSO

plot_variables(3)

plot_orientation

Specifies whether the schematic is vertical or horizontal.

TYPE

string

DEFAULT

best_fit

GROUP

plot_variables

DESCRIPTION

Specifies whether the schematic is vertical or horizontal. If you select *landscape*, the plot output is horizontal. If you select *portrait*, the plot output is vertical. If you select *best_fit* (the default), the plot output is in a manner that best fits the aspect ratio of the current design schematic.

To determine the current value of this variable, use **printvar plot_orientation**. For a list of all **plot** variables and their current values, use the **print_variable_group plot** command.

SEE ALSO

plot_variables(3)

plot_scale_factor

Specifies a scaling factor for the schematic.

TYPE

integer

DEFAULT

100

GROUP

plot_variables

DESCRIPTION

Specifies a scaling factor for the schematic. When the schematic is plotted, line widths are multiplied by the specified percentage value.

To determine the current value of this variable, use **printvar plot_scale_factor**. For a list of all **plot** variables and their current values, use the **print_variable_group plot** command.

SEE ALSO

plot_variables(3)

plot_variables

Variables that affect the **plot** command.

SYNTAX

```
Boolean plot_box = "false"
string plot_command = "lpr -Plw"
string plot_orientation = "best_fit"
integer plot_scale_factor = "100"
integer plotter_maxx = 584
integer plotter_maxy = 764
integer plotter_minx = 28
integer plotter_miny = 28
```

DESCRIPTION

These variables directly affect the **plot** command, which produces a PostScript plot of the current design. Defaults are listed above, under Syntax.

For a list of these variables and their current values, type **print_variable_group plot**. To view this manual page online, type **help plot_variables**. To view an individual variable description, type **help *var***, where *var* is the variable name.

plot_box

When *true*, a box is drawn around the plot. The default is *false*.

plot_command

Specifies the operating system command that produces a hard copy of the plot. The PostScript output of the **plot** command is piped as standard input (stdin) to the given command. If this variable is not a valid system command, an operating system error occurs during **plot**. The default is "lpr -Plw". Usually, this variable is set only once for a site in the system **.synopsys_dc.setup** initialization file.

plot_orientation

Specifies whether the schematic is vertical or horizontal. If *landscape*, the plot is output horizontally. If *portrait*, the plot is output vertically. If *best_fit* (the default value), the plot is output in a manner that best fits the aspect ratio of the current design schematic.

plot_scale_factor

Specifies a scaling factor for the schematic. Line widths are multiplied by the specified percentage value when the schematic is plotted.

plotter_maxx

Specifies the x coordinate of the upper right corner of the plot output device. Usually, this variable is set only once for a site in the **.synopsys_dc.setup** initialization file.

plotter_maxy

Specifies the y coordinate of the upper right corner of the plot output device. Usually, this variable is set only once for a site in the

.synopsys_dc.setup initialization file.

plotter_minx
: Specifies the x coordinate of the lower left corner of the plot output device. Usually, this variable is set only once for a site in the **.synopsys_setup.dc** initialization file.

plotter_miny
: Specifies the y coordinate of the lower left corner of the plot output device. Usually, this variable is set only once for a site in the **.synopsys_setup.dc** initialization file.

SEE ALSO

highlight_path (2), **plot** (2), **set_layer** (2); **schematic_variables** (3).

plotter_maxx

Specifies the x coordinate of the upper right corner of the plot output device.

TYPE

integer

DEFAULT

584

GROUP

plot_variables

DESCRIPTION

Specifies the x coordinate of the upper right corner of the plot output device. Usually, this variable is set only once for a site in the **.synopsys_dc.setup** initialization file.

To determine the current value of this variable, use **printvar plotter_maxx**. For a list of all **plot** variables and their current values, use the **print_variable_group plot** command.

SEE ALSO

plotter_maxy(3)
plotter_minx(3)
plotter_miny(3)

plotter_maxy

Specifies the y coordinate of the upper right corner of the plot output device.

TYPE

integer

DEFAULT

764

GROUP

plot_variables

DESCRIPTION

Specifies the y coordinate of the upper right corner of the plot output device. Usually, this variable is set only once for a site, in the **.synopsys_dc.setup** initialization file.

To determine the current value of this variable, use **printvar plotter_maxy**. For a list of all **plot** variables and their current values, use the **print_variable_group plot** command.

SEE ALSO

plotter_maxx(3)
plotter_minx(3)
plotter_miny(3)

plotter_minx

Specifies the x coordinate of the lower left corner of the plot output device.

TYPE

integer

DEFAULT

28

GROUP

plot_variables

DESCRIPTION

Specifies the x coordinate of the lower left corner of the plot output device. Usually, this variable is set only once for a site, in the **.synopsys_dc.setup** initialization file.

To determine the current value of this variable, use **printvar plotter_minx**. For a list of all **plot** variables and their current values, use the **print_variable_group plot** command.

SEE ALSO

plotter_maxx(3)
plotter_maxy(3)
plotter_miny(3)

plotter_miny

Specifies the y coordinate of the lower left corner of the plot output device.

TYPE

integer

DEFAULT

28

GROUP

plot_variables

DESCRIPTION

Specifies the y coordinate of the lower left corner of the plot output device. Usually, this variable is set only once for a site, in the **.synopsys_dc.setup** initialization file.

To determine the current value of this variable, use **printvar plotter_miny**. For a list of all **plot** variables and their current values, use the **print_variable_group plot** command.

SEE ALSO

plotter_maxx(3)
plotter_maxy(3)
plotter_minx(3)

port_attributes

Contains attributes that can be placed on a port.

DESCRIPTION

Contains attributes that can be placed on a port.

There are a number of commands used to set attributes. Most attributes, however, can be set with the **set_attribute** command. If the attribute definition specifies a **set** command, use it to set the attribute. Otherwise, use **set_attribute**. If an attribute is "read-only," the user cannot set it.

To determine the value of an attribute, use the **get_attribute** command. To remove attributes, use the **remove_attribute** command.

For a more detailed explanation of an attribute, refer to the manual pages of the appropriate **set** command. For information on all attributes, refer to the **attributes** manual page.

Port Attributes

connection_class

A string that specifies the connection class label to be attached to a port or to a list of ports. **compile**, **insert_pads**, and **insert_dft** will connect only those loads and drivers that have the same connection class label. The labels must match those in the library of components for the design, and must be separated by a space. The labels *universal* and *default* are reserve.d *universal* indicates that the port can connect with any other load or driver. *default* is assigned to any ports that do not have a connection class already assigned. Set with **set_connection_class**.

dont_touch_network

When a design is optimized, **compile** assigns **dont_touch** attributes to all cells and nets in the transitive fanout of **dont_touch_network** clock objects. The **dont_touch** assignment stops at the boundary of storage elements. An element is recognized as storage only if it has setup or hold constraints. Set with **set_dont_touch_network**.

driven_by_logic_one

Specifies that input ports are driven by logic one. **compile** uses this information to create smaller designs. After optimization, a port connected to logic one usually does not drive anything inside the optimized design. Set with **set_logic_one**.

driven_by_logic_zero

Specifies that input ports are driven by logic zero. **compile** uses this information to create smaller designs. After optimization, a port connected to logic zero usually does not drive anything inside the optimized design. Set with **set_logic_zero**.

driving_cell_dont_scale

When *true*, the transition time on the port using the driving cell is not scaled. Otherwise the transition time will be scaled by operating condition factors. Set with **set_driving_cell**.

driving_cell_fall
: A string that names a library cell from which to copy fall drive capability to be used in fall transition calculation for the port. Set with **set_driving_cell**.

driving_cell_from_pin_fall
: A string that names the driving_cell_fall input pin to be used to find timing arc fall drive capability. Set with **set_driving_cell**.

driving_cell_from_pin_rise
: A string that names the driving_cell_rise input pin to be used to find timing arc rise drive capability. Set with **set_driving_cell**.

driving_cell_library_fall
: A string that names the library in which to find the **driving_cell_fall**. Set with **set_driving_cell**.

driving_cell_library_rise
: A string that names the library in which to find the **driving_cell_rise**. Set with **set_driving_cell**.

driving_cell_multiply_by
: A floating point value by which to multiply the transition time of the port marked with this attribute. Set with **set_driving_cell**.

driving_cell_pin_fall
: A string that names the driving_cell_fall output pin to be used to find timing arc fall drive capability. Set with **set_driving_cell**.

driving_cell_pin_rise
: A string that names the driving_cell_rise output pin to be used to find timing arc rise drive capability. Set with **set_driving_cell**.

driving_cell_rise
: A string that names a library cell from which to copy rise drive capability to be used in rise transition calculation for the port. Set with **set_driving_cell**.

fall_delay
: Specifies an offset from the falling edge of the ideal clock waveform. Set with **set_clock_skew -fall_delay**.

fall_drive
: Specifies the drive value of high to low transition on input or inout ports. Set with **set_drive**.

fanout_load
: Specifies the fanout load on output ports. Set with **set_fanout_load**.

load
: Specifies the load value on ports. The total load on a net is the sum of all the loads on pins, ports, and wires associated with that net. Set with

set_load.

max_capacitance
A floating point number that sets the maximum capacitance value for input, output, or bidirectional ports; or for designs. The units must be consistent with those of the technology library used during optimization. Set with **set_max_capacitance**.

max_fall_delay
A floating point value that specifies the maximum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

max_fanout
Specifies the maximum fanout load for the net connected to this port. **compile** ensures that the fanout load on this net is less than the specified value. Set with **set_max_fanout**.

max_rise_delay
A floating point value that specifies the maximum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

max_time_borrow
A floating point number that establishes an upper limit for time borrowing; that is, it prevents the use of the entire pulse width for level-sensitive latches. Units are those used in the technology library. Set with **set_max_time_borrow**.

max_transition
Specifies the maximum transition time for the net connected to this port. **compile** ensures that the transition time on this net is less than the specified value. Set with **set_max_transition**.

min_capacitance
A floating point number that sets the minimum capacitance value for input or bidirectional ports. The units must be consistent with those of the technology library used during optimization. Set with **set_min_capacitance**.

min_fall_delay
A floating point value that specifies the minimum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

min_rise_delay
A floating point value that specifies the minimum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

minus_uncertainty
Specifies a negative uncertainty from the edges of the ideal clock waveform. Affects all sequential cells in the transitive fanout of this port. Set with **set_clock_skew -minus_uncertainty**.

model_drive

A non-negative floating point number that specifies the estimated drive value on ports in terms of standard drives of the current technology library. Set with **set_model_drive**.

model_load

A non-negative floating point number that specifies the estimated load value on ports in terms of standard loads of the current technology library. Set with **set_model_load**.

op_used_in_normal_op

Specifies that a scan-out port is also used in normal operation (system mode). This attribute is used by the **insert_dft** command.

output_not_used

Determines that an output port is unconnected. Used by **compile** to create smaller designs because the logic that drives an unconnected output port might not need to be maintained. After a design with an unconnected output port is compiled, the port is usually not driven by anything inside the design. Set with **set_unconnected**.

pad_location

A string value that specifies the Xilinx pad location (pin number) to be assigned to a port. Setting a **pad_location** attribute on a port causes the Synopsys XNF writer to indicate in the XNF netlist that this port has the pad location given by the value of the **pad_location** attribute. No checks are performed to verify that the specified location is valid; for valid pad locations, refer to the Xilinx *XC4000 Databook*. Set with **set_attribute**.

plus_uncertainty

Specifies a positive uncertainty from the edges of the ideal clock waveform. Affects all sequential cells in the transitive fanout of this port. Set with **set_clock_skew -plus_uncertainty**.

port_direction

Direction of a port. Value can be *in*, *out*, *inout*, or *unknown*. This attribute is read-only and cannot be set by the user.

propagated_clock

Specifies the delay of the clock edge times by propagating the values through the clock network. Affects all sequential cells in the transitive fanout of this port. If this attribute is not present, ideal clocking is assumed. Set with **set_clock_skew -propagated**.

rise_delay

Specifies an offset from the rising edge of the ideal clock waveform. Set with **set_clock_skew -rise_delay**.

rise_drive

Specifies the drive value of low to high transition on input or inout ports. Set with **set_drive**.

signal_index

Used to enumerate different ports with the same signal type (for example, scan-in ports for a design with multiple scan chains). Set with

set_signal_type.

signal_type
Used to indicate that a port is of a "special" type, such as a "clocked_on_also" port in a master/slave clocking scheme, or a "test_scan_in" pin for scan-test circuitry. Set with **set_signal_type**.

static_probability
A floating point number that specifies the percentage of time that the signal is in the logic 1 state. This information is used by **report_power**. If this attribute is not set, **report_power** will use the default value of 0.5, indicating that the signal is in the logic 1 state half the time. Set with **set_switching_activity**.

test_dont_fault
If set, ports are not faulted during test pattern generation. If no command options are specified, this attribute is set for both "stuck-at-0" and "stuck-at-1" faults. Set with **set_test_dont_fault**.

test_hold
Specifies a fixed, constant logic value at a port during test generation. Set with **set_test_hold**.

test_isolate
Indicates that the specified sequential cells, pins, or ports are to be logically isolated and considered untestable during test design rule checking by **check_test**. When this attribute is set on a cell, it is also placed on all pins of that cell. Do not set this attribute on a hierarchical cell. Use **report_test -assertions** for a report on isolated objects. Set with **set_test_isolate**.
Note: Setting this attribute suppresses the warning messages associated with the isolated objects.

toggle_rate
A positive floating point number that specifies the toggle rate; that is, the number of zero-to-one and one-to-zero transitions within a library time unit period. This information is used by **report_power**. If this attribute is not set, **report_power** will use the default value of 2*(static_probability)(1 - static_probability). The default will be scaled by any associated clock signal (if one is available). Set with **set_switching_activity**.

true_delay_case_analysis
Specifies a value to set all or part of an input vector for **report_timing -true** and **report_timing -justify**. Allowed values are *0*, *1*, *r* (rise, X to 1), and *f* (fall, X to 0). Set with **set_true_delay_case_analysis**.

SEE ALSO

get_attribute(2)
remove_attribute(2)
set_attribute(2)
attributes(3)

port_complement_naming_style

Defines the convention the **compile** command uses to rename ports complemented as a result of using the **set_boundary_optimization** command.

TYPE

string

DEFAULT

%s_BAR

GROUP

compile_variables

DESCRIPTION

Defines the convention the **compile** command uses to rename ports complemented as a result of using the **set_boundary_optimization** command.

The variable string must contain one occurrence of %s (percent s). When **compile** generates a new port name, If this does not create a unique port name within the cell, the smallest possible integer that makes the name unique is appended to the end of the name (for example, X_BAR_0, X_BAR_1, and so on).

To determine the current value of this variable, use **printvar port_complement_naming_style**. For a list of all **compile** variables and their current values, use the **print_variable_group compile** command.

SEE ALSO

compile(2)
set_boundary_optimization(2)
compile_disable_hierarchical_inverter_opt(3)
compile_variables(3)

power_attributes

Lists the predefined Power Compiler attributes.

DESCRIPTION

Attributes are properties assigned to objects such as nets, cells, pins and designs, and describe design features to be considered during optimization. A subset of attributes are specific to Power Compiler; they are only available when a Power Compiler license exists.

The Power Compiler attributes are "read-only": they cannot be set by the user. To determine the value of an attribute, use the **get_attribute** command. For information on all attributes, refer to the **attributes** manual page.

The Power Compiler attributes are grouped into the following categories:

- cell
- pin
- design

Definitions for these attributes are provided in the subsections that follow.

Cell Attributes

is_clock_gate
: *true* if the cell is a clock gate. If the clock gating logic is encapsulated in a hierarchical clock gate wrapper, this attribute will only return *true* when applied to the hierarchical instance.

is_icg
: *true* if the cell is an integrated clock gate (ICG). This attribute can only return *true* when applied to leaf cells. If the ICG cell is encapsulated in a hierarchical clock gate wrapper, this attribute will return *false* when applied to the hierarchical instance.

is_gicg
: *true* if the cell is a generic integrated clock gate (GICG). Since GICGs cannot be encapsulated in a hierarchical clock gate wrapper, this attribute can only return *true* when applied to leaf cells.

is_latch_based_clock_gate
: *true* if the cell is a latch-based clock gate.

is_latch_free_clock_gate
: *true* if the cell is a latch-free clock gate.

is_positive_edge_clock_gate
: *true* if the cell is a positive edge clock gate.

is_negative_edge_clock_gate
: *true* if the cell is a negative edge clock gate.

clock_gate_has_precontrol
true if the cell is a clock gate with (pre-latch) control point.

clock_gate_has_postcontrol
true if the cell is a clock gate with (post-latch) control point.

clock_gate_has_observation
true if the cell is a clock gate with observation point.

is_clock_gated
true if the cell is a clock gated register or clock gate.

clock_gating_depth
The number of clock gates on the clock path to this cell; -1 if the cell is not a clock gate or register.

clock_gate_level
The number of clock gates on the longest clock branch in the fanout of this cell; -1 if not a clock gate.

clock_gate_fanout
The number of registers and clock gates in the direct fanout of the clock gate; -1 if not a clock gate.

clock_gate_register_fanout
The number of registers in the direct fanout of the clock gate; -1 if not a clock gate.

clock_gate_multi_stage_fanout
The number of clock gates in the direct fanout of the clock gate; -1 if not a clock gate.

clock_gate_transitive_register_fanout
The number of register in the transitive fanout of the clock gate; -1 if not a clock gate.

clock_gate_module_fanout
The number of modules in the local fanout of the clock gate; -1 if not a clock gate.

is_operand_isolator
true if the cell is an operand isolation cell.

is_isolated_operator
true if the cell is an operator that was isolated with operand isolation.

operand_isolation_style
Stores the operand isolation style of the isolation cell or isolated operator.

Pin Attributes

is_clock_gate_enable_pin
true if the pin is a clock gate enable input.

is_clock_gate_clock_pin
true if the pin is a clock gate clock input.

is_clock_gate_output_pin
true if the pin is a gated-clock output of a clock gate.

is_clock_gate_test_pin
true if the pin is a clock gate scan-enable or test-mode input.

is_clock_gate_observation_pin
true if the pin is a clock gate observation point.

is_operand_isolation_control_pin
true if the pin is the control pin of an operand isolation cell.

is_operand_isolation_data_pin
true if the pin is the data input of an operand isolation cell.

is_operand_isolation_output_pin
true if the pin is the data output of an operand isolation cell.

"Design Attributes

is_clock_gating_design
true if the design is a clock gating design.

is_clock_gating_observability_design
true if the design is a clock gating observability design.

SEE ALSO

get_attribute(2)
attributes(3)

power_cg_all_registers

Specifies to the **insert_clock_gating** command whether to clock gate all registers, including those that do not meet the necessary requirements.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

Specifies whether to clock gate all registers, including those that do not meet the necessary requirements. If this variable is set to *false* (the default value), registers that do not meet the setup, width or enable condition are not considered for clock gating, unless they are explicitly included with the **set_clock_gating_registers** command. When set to *true*, a redundant clock gate will be inserted for these registers. This can be useful for clock tree balancing. If necessary, the redundant clock gates will be duplicated to meet the max_fanout constraint. Note that the minimum_bitwidth constraint is not honored for the redundant clock gated registers.

This variable is valid only with Design Compiler (dc_shell, dc_shell-t) and Physical Compiler (psyn_shell).

To determine the current value of this variable, type **printvar power_cg_all_registers**. For a list of power variables and their current values, type **print_variable_group power**.

SEE ALSO

insert_clock_gating(2)
elaborate(2)
power_variables(3)

power_cg_auto_identify

Activates automatic identification of Power Compiler inserted clock gating circuitry from a structural netlist.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

When this variable is set to **true**, Power Compiler inserted clock gates are automatically identified. Identification refers to the process of detecting clock gates and the corresponding gated element association, and setting different attributes on these objects. These attributes enable the subsequent steps in the tool to work efficiently.

Clock gate identification is required when a netlist is passed in ASCII format. This is not required if a binary database (such as DDC or MilkyWay) is used to transfer design data between tool invocations.

The variable only works in **dc_shell**.

By using the automatic identification method, each time a command that works on clock gating circuitry is called identification is performed. This assures that the command can make use of the current clock gating configuration of the design.

SEE ALSO

compile_ultra(2)
identify_clock_gating(2)
insert_clock_gating(2)
power_variables(3)

power_cg_balance_stages

Controls clock gate stage balancing is on or off during **compile [-incremental_mapping] -gate_clock** or **compile_ultra [-incremental_mapping] -gate_clock**.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

This variable can be used to reduce the un-evenness in the number of stages of clock gates feeding different register banks. If this variable is set to true, **compile -gate_clock** or **compile_ultra -gate_clock** would reconfigure the different stage of clock gates so that there is exactly the same number of clock gates in each path from the clock root to the clock pin of all *gatable* register banks. It is necessary to define clocks using **create_clock** command for each clock network where clock gates need to be reconfigured.

Only the tool inserted clock gates and integrated clock gating (ICG) cells are considered during stage balancing.

To determine the current value of this variable, type **printvar power_cg_balance_stages**.

SEE ALSO

set_clock_gating_style(2)
compile(2)
compile_ultra(2)
power_cg_reconfig_stages(3)
power_variables(3)

power_cg_cell_naming_style

Specifies the naming style for clock gating cells created during **insert_clock_gating**.

TYPE

String

DEFAULT

""

GROUP

power

DESCRIPTION

Determines the way clock gating cells are named.

This variable must be set before issuing **insert_clock_gating** command.

This variable can contain any string in addition to the different tokens listed below. The tokens are replaced by appropriate value during clock gating and other strings are retained the way they are.

```
set power_cg_cell_naming_style \
   "prefix_%c_%e_midfix_%r_%R_%d_suffix"
where,
     clockgates and not applicable for factored clock gates)
     for module or factored clock gates)
```

For all object names only simple (non-hierarchical) names are used.

If there is no occurrence of "%d" in the power_cg_cell_naming_style, Power Compiler will assume a %d at the end.

Example :

```
   dc_shell-t> set power_cg_cell_naming_style "clk_gate_%r_%d
   dc_shell-t> set power_cg_cell_naming_style "clock_gate_cell_%c_%r_%d_name
```

SEE ALSO

insert_clock_gating(2)
power_cg_module_naming_style(3)
power_cg_gated_clock_net_naming_style(3)

power_cg_derive_related_clock

When *true*, clock domain relationship between registers will be derived from the hierarchical context.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

For latch-free clock gate insertion with the **insert_clock_gating** command the setup condition must be met before a bank can be clock gated. This implies that the enable condition of the bank must be combinationally dependent on nets that are known to be synchronous (i.e., belong to the same clock domain) with the registers of the bank that are being clock gated.

The clock domain of nets is determined by the registers withing the module (subdesign) that drive the nets and the clock relationship specified for ports with the **set_input_delay** command. By default the clock domain relationship is analyzed locally, i.e., within the module. This is to ensure that clock gating on subdesigns is context independent: if a module is instantiated from a design different than the current design, the clock gating result will still be correct.

If the non-combinational driver (register or top level port) of an input to a module exists outside of that module, the clock domain of that input is considered unknown and no latch-free clock gating can be performed with enable conditions that depend on that particular input.

The setup condition can be relaxed to perform context specific analysis of the clock domain relationship. This is achieved by setting the variable **power_cg_derive_related_clock** to *true*. In that case the clock domain of any net will be derived from the non-combinational drivers in the (hierarchical) fanin of the net.

To determine the current value of this variable, type **printvar power_cg_derive_related_clock**. For a list of power variables and their current values, type **print_variable_group power**.

SEE ALSO

insert_clock_gating(2)
set_clock_gating_style(2)

power_variables(3)

power_cg_designware

Performs clock gating on DesignWare sequential components in the design.

The use of **power_cg_designware** variable will be obsolete in a future release. Clock gating insertion with **compile_ultra -gate_clock** automatically inserts clock gates in DesignWare modules.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

The use of the **power_cg_designware** variable will be obsolete in a future release. Clock gating insertion using the **compile_ultra** command with the **-gate_clock** option automatically inserts clock gates in DesignWare modules.

The **power_cg_designware** variable instructs the **compile** command to invoke DesignWare clock gating. During the compile stage, all DesignWare sequential components are clock gated if there is a clock gating opportunity.

A **Clock Gate Insertion Report** is generated after **Implementation Selection**. You can use the **insert_dft** command to connect test ports inside DesignWare components. The **report_clock_gating** command can be used to get more details about the clock gating result.

The following example shows a TCL script invoking DesignWare clock gating:

```
   prompt> set power_cg_designware true
   prompt> analyze -f verilog DW_cntr_gray_inst.v
   prompt> elaborate DW_cntr_gray_inst
   prompt> create_clock -period 5 [get_ports inst_clk]
   prompt> compile
   prompt> report_clock_gating
```

SEE ALSO

compile(2)
insert_clock_gating(2)
set_clock_gating_style(2)

power_cg_enable_alternative_algorithm

Specifies to the **insert_clock_gating**, **compile-gate_clock** and **compile_ultra -gate_clock** commands whether to use an alternative algorithm to find gatable registers.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

Depending on the design clock gating can have a long runtime. When set to *true* **power_cg_enable_alternative_algorithm** changes the clock gating algorithm in Design Compiler. The alternative algorithm that is enabled often reduces the runtime for clock gating especially for larger designs, but might increase the number of clock gates that are created compared to the default algorithm. The number of gated registers may vary.

To determine the current value of this variable, type **printvar power_cg_enable_alternative_algorithm**.

SEE ALSO

insert_clock_gating(2)
compile_ultra(2)
compile(2)
power_variables(3)

power_cg_flatten

Specifies to different **ungroup** commands whether to flatten Synopsys clock-gating cells.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

Specifies to differnt **ungroup** commands whether to flatten Synopsys clock-gating cells. The list of commands include **ungroup**, **compile -ungroup**, **optimiz_regsiters -ungroup**, **balance_registers** and **ungroup**.

If this variable is set to *false* (the default value), the clock-gating cells are not flattened during any ungroup step. To flatten the clock gating cells, set the value to *true* before using the **ungroup** command or any other ungrouping steps listed above.

In normal usage, ungrouping the clock gates is not recommended. Ungrouping the clock gates before compile could have serious side effects. For example, ungrouped clock gates can not be mapped to integrated clock gating cells. Power Compiler commands like **report_clock_gating**, **remove_clock_gating** and **rewire_clock_gating** assumes that the clock gates have a hierarchy of its own. Also Physical Compiler placement of clock gates and registers may not be optimal when clock gating cells are flattened. Flattened clock gates are supported when using integrated clock gating cells, provided the flattening is done only after **compile**.

This variable is valid only with Design Compiler (dc_shell, dc_shell-t) and Physical Compiler (psyn_shell).

To determine the current value of this variable, type **printvar power_cg_flatten**. For a list of power variables and their current values, type **print_variable_group power**.

SEE ALSO

compile(2)
insert_clock_gating(2)
set_clock_gating_style(2)
ungroup(2)
power_variables(3)

power_cg_gated_clock_net_naming_style

Specifies the naming style for gated clock nets created during **insert_clock_gating**.

TYPE

String

DEFAULT

""

GROUP

power

DESCRIPTION

Determines the way gated clock nets are named.

This variable must be set before issuing **insert_clock_gating** command.

This variable can contain any string in addition to the different tokens listed below. The tokens are replaced by appropriate value during clock gating.

```
set power_cg_gated_clock_net_naming_style \
   "prefix_%c_%n_%g_%d_suffix'
```

For all object names only simple (non-hierarchical) names are used.

If there is no occurrence of "%d" in the power_cg_gated_clock_net_naming_style, Power Compiler will assume a %d at the end.

Example :

```
   dc_shell-t> set power_cg_gated_clock_net_naming_style "gated_%c_%d"
```

SEE ALSO

insert_clock_gating(2)
power_cg_module_naming_style(3)
power_cg_cell_naming_style(3)
power_cg_gated_clock_net_naming_style(3)

power_cg_ignore_setup_condition

When *true*, the setup condition will be ignored for latch-free clock gating.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

For latch-free clock gate insertion with the **insert_clock_gating** command the setup condition must be met before a bank can be clock gated. This implies that the enable condition of the bank must be combinationally dependent on nets that are known to be synchronous (i.e., belong to the same clock domain) with the registers of the bank that are being clock gated.

The setup condition can be ignored during latch-free clock gating. This is achieved by setting the variable **power_cg_ignore_setup_condition** to *true*. Note that this is generally not safe, since latch-free clock gating of a bank with an enable that is not synchronous to the registers of that bank can lead to undesired glitches on the clock net. In this case it is the responsability of the designer to ensure that the result after clock gating is functionally correct.

To determine the current value of this variable, type **printvar power_cg_ignore_setup_condition**. For a list of power variables and their current values, type **print_variable_group power**.

SEE ALSO

set_clock_gating_style(2)
insert_clock_gating(2)
power_variables(3)

power_cg_inherit_timing_exceptions

Specifies that during **compile -gate_clock** or **compile_ultra [-incr] -gate_clock**, timing exceptions defined on registers have to be automatically inferred on to the enable pin of the clock gate that is gating these registers.

TYPE

Boolean

DEFAULT

false

GROUP

power

DESCRIPTION

If this variable is set before **compile -gate_clock** or **compile_ultra -gate_clock**, then clock gate insertion tries to inherit the timing exceptions of the gated registers to clock gate enable pin. The following conditions apply.

Only cell level exceptions are inherited. This means all synchronous pins of a register should have the same set of timing exceptions before this can be inferred onto a clock gate.

If a set of registers with same enable condition has different timing exceptions among them, then this is split into smaller set of registers with homogeneous exceptions before clock gate insertion. This may create small banks that fall below the minimum bit width specified for clock gating and hence may remain ungated.

SEE ALSO

compile_ultra(2)
compile(2)

power_cg_module_naming_style

Specifies the naming style for clock gating modules created during **insert_clock_gating**.

TYPE

String

DEFAULT

""

GROUP

power

DESCRIPTION

Determines the way clock gate modules (hierarchical designs created for clock gates during **insert_clock_gating**) are named.

This variable must be set before issuing **insert_clock_gating** command.

This variable can contain any string in addition to the different tokens listed below. The tokens are replaced by appropriate value during clock gating.

```
set power_cg_module_naming_style \
   "prefix_%e_%l_midfix_%p_%t_%d_suffix"
where,
prefix/midfix/suffix are just examples of any constant strings
that can be specified.
     concatenated target_library names
```

If there is no occurrence of "%d" in the power_cg_module_naming_style, Power Compiler will assume a %d at the end.

Example :

```
   dc_shell-t> set power_cg_module_naming_style "SNPS_CLOCK_GATE_%e_%p
   dc_shell-t> set power_cg_module_naming_style "clock_gate_module_%e_%t_%d
```

SEE ALSO

insert_clock_gating(2)
power_cg_cell_naming_style(3)
power_cg_gated_clock_net_naming_style(3)

power_cg_print_enable_conditions

When *true*, the enable conditions of registers and clock gates will be reported during clock gate insertion.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

For debugging purposes and to achieve a general better understanding of the design structure and functionality it can be useful to print the enable conditions of clock gates and registers when performing automatic RTL clock gate insertion with the **insert_clock_gating** command.

The enable condition of a register or clock gate is a combinational function of nets in the design. As such, enable conditions can be represented by Boolean expressions of nets. The enable condition of a register represents the states for which a clock signal must be passed to the register. The enable condition of a clock gate corresponds to the states for which a clock will be passed to the registers in the fanout of the clock gate. Power Compiler utilizes the enable condition of the registers for clock gate insertion.

Reporting of the enable conditions during clock gating can be enabled by setting the variable **power_cg_print_enable_conditions** to *true* before issuing the **insert_clock_gating** command.

To determine the current value of this variable, type **printvar power_cg_print_enable_conditions**. For a list of power variables and their current values, type **print_variable_group power**.

SEE ALSO

insert_clock_gating(2)
power_cg_print_enable_conditions_max_terms(3)
power_variables(3)

power_cg_print_enable_conditions_max_terms

Specifies the maximum number of product terms to be reported in the sum of product expansion of the enable condition.

TYPE

Integer

DEFAULT

10

GROUP

power_variables

DESCRIPTION

For debugging purposes and to achieve a general better understanding of the design structure and functionality it can be useful to print the enable conditions of clock gates and registers when performing automatic RTL clock gate insertion with the **insert_clock_gating** command.

The enable conditions are reported as a sum of product expression. Since for complex expressions such a representation is known to grow very large, the maximum number of product terms is limited by the variable **power_cg_print_enable_conditions_max_terms**. By default at most 10 product terms are printed. If the actual number of product terms exceeds this limit, the enable condition is reported as "?? (too many product terms)". If necessary, the number of product terms to be shown can be increased by setting the variable **power_cg_print_enable_conditions_max_terms** to a larger value.

To determine the current value of this variable, type **printvar power_cg_print_enable_conditions_max_terms**. For a list of power variables and their current values, type **print_variable_group power**.

SEE ALSO

insert_clock_gating(2)
power_cg_print_enable_conditions(3)
power_variables(3)

power_cg_reconfig_stages

Controls the reconfiguration of multistage clock gates during **compile [-incremental_mapping] -gate_clock** or **compile_ultra [-incremental_mapping] -gate_clock**.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

Multistage reconfiguration involves addition or reduction of stages of clock gates. You specify the maximum permissible number of stages of clock gates in the design using the **set_clock_gating_style -num_stages** command. A clock gate stage is added when a common enable condition can be factored out of many clock gates without violating the maximum stages specified. A clock gate stage is reduced if the design has more stages than what is specified.

These reconfigurations are performed during **compile_ultra -gate_clock** or **compile -gate_clock**, **only** if the variable **power_cg_reconfig_stages** is set to true. Also it is necessary to define clocks using **create_clock** command for each clock network where clock gates need to be reconfigured.

Only the tool inserted clock gates and integrated clock gating (ICG) cells can be reconfigured.

To determine the current value of this variable, type **printvar power_cg_reconfig_stages**.

SEE ALSO

set_clock_gating_style(2)
compile(2)
compile_ultra(2)
power_cg_balance_stages(3)
power_variables(3)

power_default_static_probability

Specifies the default static probability value.

TYPE

Float

DEFAULT

0.5

GROUP

power_variables

DESCRIPTION

The variables:

power_default_static_probability power_default_toggle_rate
power_default_toggle_rate_type

are used to determine the switching activity of non-user annotated nets that are driven by primary inputs or black-box cells. For other unannoated nets, Power Compiler will propagate the switching activities of the driving cell inputs based on the cell functionality to derive the switching activity required for power calculations. This mechanism cannot be used for primary inputs and black-box outputs. Instead the following values are used for these type of nets:

- User annotated values are used, even when the net is partially annotated (for example, the static probability is annotated, but the toggle rate is not).

- In some cases, unannotated switching activity values may still be accurately derived, for example, if the net drives a buffer cell and the output of this cell is user annotated, then the user annotated values are used as the default values. Also, if the input is a clock then the clock period and waveform are used to derive the switching activity values.

- If the static probability is not annotated then the value of the **power_default_static_probability** variable is used for the static probability value.

- If the toggle rate is not annotated, then the default toggle rate value is derived from the **power_default_toggle_rate_type** and **power_default_toggle_rate** values. If the value of the **power_default_toggle_rate_type** variable is *fastest_clock* then the following is used for the toggle rate value:

dtr * fclk

where fclk is the frequency of the related clock if specified by the **set_switching_activity** command, or the frequency of the fastest clock in the design.

If the design has no clocks, then a value of 1.0 is used for fclk; and dtr is the value of the **power_default_toggle_rate** variable.

If the value of **power_default_toggle_rate_type** is *absolute*, then the value of the **power_default_toggle_rate** variable is used as the toggle rate.

The value of **power_default_static_probability** should be between 0.0 and 1.0, both inclusive. The value of **power_default_toggle_rate** should be greater or equal to 0.0. Also, if the value of **power_default_static_probability** is 0.0 or 1.0, then the value of **power_default_toggle_rate** should be 0.0. If the value of **power_default_toggle_rate** is 0.0, then the value of **power_default_static_probability** should be either 0.0 or 1.0.

The default value of power_default_toggle_rate variable is 0.1. The default value of power_default_static_probability variable is 0.5.

The value of **power_default_toggle_rate_type** can be either *fastest_clock* or *absolute*. The default value is *fastest_clock*.

SEE ALSO

set_switching_activity(2)
power_default_toggle_rate(3)
power_default_toggle_rate_type(3)

power_default_toggle_rate

Specifies the default toggle rate value.

TYPE

Float

DEFAULT

0.1

GROUP

power_variables

DESCRIPTION

The variables:

power_default_static_probability power_default_toggle_rate
power_default_toggle_rate_type

are used to determine the switching activity of non-user annotated nets that are driven by primary inputs or black-box cells. For other unannoated nets, Power Compiler will propagate the switching activities of the driving cell inputs based on the cell functionality to derive the switching activity required for power calculations. This mechanism cannot be used for primary inputs and black-box outputs. Instead the following values are used for these type of nets:

- User annotated values are used, even when the net is partially annotated (for example, the static probability is annotated, but the toggle rate is not).

- In some cases, unannotated switching activity values may still be accurately derived, for example, if the net drives a buffer cell and the output of this cell is user annotated, then the user annotated values are used as the default values. Also, if the input is a clock then the clock period and waveform are used to derive the switching activity values.

- If the static probability is not annotated then the value of the **power_default_static_probability** variable is used for the static probability value.

- If the toggle rate is not annotated, then the default toggle rate value is derived from the **power_default_toggle_rate_type** and **power_default_toggle_rate** values. If the value of the **power_default_toggle_rate_type** variable is *fastest_clock* then the following is used for the toggle rate value:

dtr * fclk

where fclk is the frequency of the related clock if specified by the **set_switching_activity** command, or the frequency of the fastest clock in the design.

If the design has no clocks, then a value of 1.0 is used for fclk; and dtr is the value of the **power_default_toggle_rate** variable.

If the value of **power_default_toggle_rate_type** is *absolute*, then the value of the **power_default_toggle_rate** variable is used as the toggle rate.

The value of **power_default_static_probability** should be between 0.0 and 1.0, both inclusive. The value of **power_default_toggle_rate** should be greater or equal to 0.0. Also, if the value of **power_default_static_probability** is 0.0 or 1.0, then the value of **power_default_toggle_rate** should be 0.0. If the value of **power_default_toggle_rate** is 0.0, then the value of **power_default_static_probability** should be either 0.0 or 1.0.

The default value of power_default_toggle_rate variable is 0.1. The default value of power_default_static_probability variable is 0.5.

The value of **power_default_toggle_rate_type** can be either *fastest_clock* or *absolute*. The default value is *fastest_clock*.

SEE ALSO

set_switching_activity(2)
power_default_static_probability(3)
power_default_toggle_rate_type(3)

power_default_toggle_rate_type

Specifies the default toggle rate type.

TYPE

String

DEFAULT

fastest_clock

GROUP

power_variables

DESCRIPTION

The variables:

power_default_static_probability power_default_toggle_rate
power_default_toggle_rate_type

are used to determine the switching activity of non-user annotated nets that are driven by primary inputs or black-box cells. For other unannoated nets, Power Compiler will propagate the switching activities of the driving cell inputs based on the cell functionality to derive the switching activity required for power calculations. This mechanism cannot be used for primary inputs and black-box outputs. Instead the following values are used for these type of nets:

- User annotated values are used, even when the net is partially annotated (for example, the static probability is annotated, but the toggle rate is not).

- In some cases, unannotated switching activity values may still be accurately derived, for example, if the net drives a buffer cell and the output of this cell is user annotated, then the user annotated values are used as the default values. Also, if the input is a clock then the clock period and waveform are used to derive the switching activity values.

- If the static probability is not annotated then the value of the **power_default_static_probability** variable is used for the static probability value.

- If the toggle rate is not annotated, then the default toggle rate value is derived from the **power_default_toggle_rate_type** and **power_default_toggle_rate** values. If the value of the **power_default_toggle_rate_type** variable is *fastest_clock* then the following is used for the toggle rate value:

dtr * fclk

where fclk is the frequency of the related clock if specified by the **set_switching_activity** command, or the frequency of the fastest clock in the design.

If the design has no clocks, then a value of 1.0 is used for fclk; and dtr is the value of the **power_default_toggle_rate** variable.

If the value of **power_default_toggle_rate_type** is *absolute*, then the value of the **power_default_toggle_rate** variable is used as the toggle rate.

The value of **power_default_static_probability** should be between 0.0 and 1.0, both inclusive. The value of **power_default_toggle_rate** should be greater or equal to 0.0. Also, if the value of **power_default_static_probability** is 0.0 or 1.0, then the value of **power_default_toggle_rate** should be 0.0. If the value of **power_default_toggle_rate** is 0.0, then the value of **power_default_static_probability** should be either 0.0 or 1.0.

The default value of power_default_toggle_rate variable is 0.1. The default value of power_default_static_probability variable is 0.5.

The value of **power_default_toggle_rate_type** can be either *fastest_clock* or *absolute*. The default value is *fastest_clock*.

SEE ALSO

set_switching_activity(2)
power_default_static_probability(3)
power_default_toggle_rate(3)

power_do_not_size_icg_cells

Controls whether compile does not size the integrated clock-gating cells in a design to correct DRC violations because doing so may result in lower area and power.

TYPE

Boolean

DEFAULT

true

GROUP

power_variables

DESCRIPTION

When this variable is set to **true**, **compile** does not size the integrated clock-gating cells in the design to correct DRC violations, because doing so may result in lower area and power when the integrated clock-gating cell is the last element in the clock tree and drives all gated registers.

The default is **true**.

If the clock tree synthesis (CTS) tool inserts buffers after the clock gating, the fanout of the integrated clock-gating cell is limited to the clock-tree buffers. While running **compile** before performing clock tree synthesis, this information is not available in the design. If you set **power_do_not_size_icg_cells** to **true**, **compile** ignores DRC violations for the integrated clock-gating cells, because the CTS tool would insert buffers at the output of the cell. Once your design netlist has CTS buffers, you can set this variable to **false** to enable **compile** to fix any DRC violation still existing for the integrated clock-gating cell.

This variable is valid only under dc_shell (Design Compiler).

To determine the current value of this variable, type **printvar power_do_not_size_icg_cells**. For a list of power variables and their current values, type **print_variable_group power**.

SEE ALSO

compile(2)
elaborate(2)
set_clock_gating_style(2)

power_driven_clock_gating

Controls whether switching activity and dynamic power of the register banks should be considered when optimizing the clock gating of the design.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

When the **-gate_clock** option is specified with **compile** or **compile_ultra**, clock gates are automatically inserted or removed during the optimization phase. By default this optimization is not based on the switching activity and dynamic power of the register banks.

In general, power-driven clock gating will yield better results at the expense of increased runtime. However, in some cases it may be best to not use the power-driven aspect of the optimization algorithm because the power calculation is expected not to be sufficiently accurate. This could be because either no realistic switching activity can be provided or because the target library does not have power characterization for all cells.

In the default case, when power-driven clock gating is switched off, clock gates will be inserted if the register bank size meets the minimum bit-width condition set by the **set_clock_gating_style** command. This is similar to the behavior of the **insert_clock_gating** command, except that clock gate insertion can be performed on a mapped netlist (i.e., gate-level clock gating).

To enable power-driven gate-level clock gating, set the variable **power_driven_clock_gating** to *true*.

To determine the current value of this variable, type **printvar power_driven_clock_gating**. For a list of power variables and their current values, type **print_variable_group power**.

SEE ALSO

set_clock_gating_style(2)
compile(2)
compile_ultra(2)
power_variables(3)

power_enable_one_pass_power_gating

When *true*, one-pass flow power gating will be enabled.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

When *true*, one-pass flow power gating will be enabled. One pass flow eliminates the need of compile incremental, and simplifies the user interface.

The default is *false*.

The following are examples of the original power gating flow and the one-pass flow.

- Original power gating flow:

```
set_power_gating_style -type CLK_FREE [get_cells lev1b_inst/*reg*]
set_power_gating_signal -power_pin_index 1 [get_pin lev1b_inst/retain]
set_power_gating_signal -power_pin_index 2 [get_pin lev1b_inst/shutdown]
set power_enable_power_gating true
compile
hookup_power_gating_ports
compile -incr
write -format verilog -hierarchy -output post_compile.v
```

- One-pass power gating flow:

```
set_power_gating_style -type CLK_FREE [get_cells lev1b_inst/*reg*]
set_power_gating_signal -power_pin_index 1 [get_pin lev1b_inst/retain]
set_power_gating_signal -power_pin_index 2 [get_pin lev1b_inst/shutdown]
set power_enable_one_pass_power_gating true
hookup_power_gating_ports -port_naming_style \
      {lev1a_inst/retain lev1a_inst/shutdown} \
      -default_port_naming_style power_pin_%d
compile
write -format verilog -hierarchy -output post_compile.v
```

SEE ALSO

```
hookup_power_gating_ports(2)
set_power_gating_signal(2)
set_power_gating_style(2)
```

power_enable_power_gating

When set to true **compile** will enable the power gating flow which allows the selected retention registers from target library to be used to map sequential elements.

TYPE

Boolean

DEFAULT

false

GROUP

power

DESCRIPTION

The **power_enable_power_gating** variable enables and disables power gating flow during the **compile** and **physopt** commands.

Retention registers are the registers which can save the values of the registers and restore it back later. Retention registers might have different styles. The cell level attribute *power_gating_cell* in target libraries are used to specify the styles. In order to only allow the retention registers with certain types used for the specified sequential elements during **compile**, the **power_enable_power_gating** variable needs to be set to *true*. The command requires a Power Compiler license during **compile**, otherwise Design Compiler will not be able to handle retention register correctly. If the variable is *true*, it will also prevent **compile** and **physopt** commands to swap the retention registers back to regular sequential elements.

SEE ALSO

compile(2)

power_fix_sdpd_annotation

Specifies whether user-annotated SDPD switching activity annotation is corrected before it is used.

TYPE

Boolean

DEFAULT

true

GROUP

power_variables

DESCRIPTION

This variable specifies whether Power Compiler will modify the user-annotated state-dependent and/or path-dependent (SDPD) switching activity to fix any inconsistencies before they are used. The accuracy of switching activity annotation, including the SDPD annotation, affects the accuracy of power calculation, and when this variable is set Power Compiler will check the SDPD annotation for inconsistencies. The SDPD annotation will be modified automatically to fix any inconsistencies found. In most cases, some inconsistencies are created during normal switching activity flows, and the SDPD fixing step modifies the SDPD annotation slightly to improve the power estimation accuracy. For example, during SAIF generation using a simulator, the total of rise and fall toggle on a pin may be different, if say, an odd number of toggles are captured. In this case, the SDPD fixing step scales the SDPD toggle rate annotations so that the rise and fall totals are the same. Setting this variable to false will disable the SDPD fixing step. Setting the variable **power_fix_sdpd_annotation_verbose** to true makes the SDPD fixing step issue verbose messages when user annotated switching activity is modified.

The following checks and modifications are performed during the SDPD fixing step:
• false states (states that always evaluate to 0, including the default state on cells whose other states cover all possible states) with non-zero state-dependent (SD) static probabilities have their static probability set to 0.0.

• Unannotated false states on cells with partially annotated SD static probabilities are automatically annotated with the 0.0.

• Cells with fully annotated SD static probabilities have their static probabilities scaled so that they add up to 1.0.

• Cells with partially annotated SD static probabilities that add up to more than 1.0 have their static probabilities scaled down so that they add up to 1.0. On such cells, an SD static probability of 0.0 is set on the unannotated states.

• false states/arcs with non-zero state-dependent and/or path-dependent (SDPD) toggle rates have their toggle rate set to 0.0.

• Unannotated false states/arcs on pins with partially annotated SDPD toggle rates are automatically annotated with the 0.0.

• Pins with fully annotated SDPD toggle rates have their toggle rates scaled so that they add up to the non-SDPD toggle rate of the pin (that is, the non-SDPD toggle rate on the net connected to the pin).

• Pins with partially annotated SDPD toggle rates that add up to more than the pin toggle rate have their SDPD toggle rates scaled down so that they add up to the non-SDPD toggle rate. On such pin, an SDPD toggle rate of 0.0 is set on the unannotated states/arcs.

• Pins with partially annotated SDPD toggle rates have their toggle rates scaled so that the totals of rise and fall SDPD toggle rates are equal.

• The sum of the SDPD toggle rates on pins with fully annotated SDPD toggle rate information is annotated as the non-SDPD pin toggle rate (that is, the total toggle rate of the net connected to the pin) if this was not previously annotated.

SEE ALSO

power_fix_sdpd_annotation_verbose(3)
power_sdpd_message_tolerance(3)

power_fix_sdpd_annotation_verbose

Specifies whether verbose messages are reported during fixing of user-annotated SDPD switching activity.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

If this variable is set to true, then an information or warning message is reported for every modification (exceeding a tolerance criteria) to the user-annotated SDPD switching activity performed by the SDPD fixing step. The SDPD fixing step is enabled by the **power_fix_sdpd_annotation** variable; please check the man page of **power_fix_sdpd_annotation** for more information on this step. When the **power_fix_sdpd_annotation_verbose** variable is set to false, such verbose message are not reported and a single message indicating that SDPD annotation is being modified is reported instead. Whether this variable is set to true or not, SDPD fixing messages are only reported if they exceed the tolerance criteria specified by the **power_sdpd_message_tolerance** variable.

SEE ALSO

power_fix_sdpd_annotation(3)
power_sdpd_message_tolerance(3)

power_hdlc_do_not_split_cg_cells

When *true*, finsert_clock_gating does not split clock-gating cells to limit their fanout.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

When *true*, **insert_clock_gating** does not split clock-gating cells to limit their fanout. When false (the default), **insert_clock_gating** splits clock-gating cells to limit their fanout. This activity is based on the value specified by the **set_clock_gating_style -max_fanout** command, whose default is unlimited. When *true*, **insert_clock_gating** does not split clock-gating cells, resulting in a netlist where all registers are gated by a single clock-gating cell if they share the same enable signal. And, if true, **insert_clock_gating** does not honor the value specified by **set_clock_gating_style -max_fanout**.

Set the variable to *true* if your clock-tree synthesis (CTS) tool inserts buffers after the clock-gating cell. In this case, the fanout of the clock-gating cell is limited to the buffers inserted by the CTS tool. This information is not available in the design while the **insert_clock_gating** command is performing RTL clock gating. If you set this variable to *true*, **insert_clock_gating** does not split the load of the clock-gating cells (by duplicating the cells) to save area and power.

This variable is valid under dc_shell (Design Compiler) and HDL Compiler.

To determine the current value of this variable, type **printvar power_hdlc_do_not_split_cg_cells**. For a list of power variables and their current values, type **print_variable_group power**.

SEE ALSO

compile(2)
insert_clock_gating(2)
set_clock_gating_style(2)

power_keep_license_after_power_commands

Affects the amount of time a Power Compiler license is checked out during a dc_shell (Design Compiler) session.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

This variable affects the amount of time a Power Compiler license is checked out during a dc_shell (Design Compiler) session.

When **true**, a Power Compiler license that is checked out under dc_shell remains checked out throughout the dc_shell session. When this variable is set to **false** (the default value), the Power Compiler license remains checked out only as long as a command is using it, and at the completion of the command, the license is released.

To determine the current value of this variable, type **printvar power_keep_license_after_power_commands**. For a list of power variables and their current values, type **print_variable_group power**.

SEE ALSO

report_power(2)

power_lib2saif_rise_fall_pd

Specifies whether lib2saif generates forward SAIF files with directives to generate rise/fall dependent path-dependent toggle counts.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

When this variable is set to true the **lib2saif** command generates forward SAIF files with directives to generate separate rise and fall values for non-state-dependent path-dependent (PD) toggle counts. When this variable is set to false, directives to generate just the total (of the rise and fall) values for non-state-dependent PD toggle counts are generated. For more accurate power calculations, it is suggested to use separate rise and fall toggle counts, however older simulators and simulation interfaces may not recognize such directives and will fail to read the library forward SAIF file. The Synopsys SAIF generation PLI utility provided with Power Compiler X-2005.09 supports directives for separate rise and fall values for PD toggle rates, but older versions of the utility do not. The default value of this variable is currently false for backward compatibility, but will be changed to true in future releases.

Note that this variable affects only directives for path-dependent toggle counts that are not state-dependent. Whether this variable is set to true or false, directives for separate rise and fall values are generated for state-dependent and both state and path-dependent toggle counts.

SEE ALSO

lib2saif(2)

power_min_internal_power_threshold

Specifies the minimum cell internal power value that can be used in power calculations.

TYPE

string

DEFAULT

""

GROUP

power_variables

DESCRIPTION

The value of this variable specifies the minimum threshold used for the cell internal power value. Internal power values are computed using the cell’s switching activity and internal power characterization. If this variable has a numeric value and a cell’s computed internal power is less than the variable’s value, then the variable’s value is used instead. Note that this variable has an effect only if it has a numeric value. The default value of this variable is "" which is non-numeric and therefore specifies that no minimum threshold is used on the cell internal power. In general, this variable should not be used since the internal power characterization specifies the correct internal power values.

power_model_preference

Specifies the preference between the CCS power and the NLPM models in library cells that have power specified in both models.

TYPE

string

DEFAULT

nlpm

GROUP

power_variables

DESCRIPTION

A library can contain CCS power, NLPM or both types of data within a cell definition. Use this variable to specify the power model preference if the library contains both NLPM and CCS power data.

Allowed values are as follows:
ccs instructs Power Compiler to use CCS power data in the library (if present) to calculate both static and dynamic power. If CCS power data is not found, Power Compiler uses NLPM data.
nlpm (the default) instructs Power Compiler to use NLPM data. If NLPM data is not found, Power Compiler uses CCS power data.

If neither CCS power nor NLPM data is found for a cell in the library, this cell is not characterized for power analysis.

power_opto_extra_high_dynamic_power_effort

This variable makes the **compile** command invoke more dyanmic power optimization algorithms.

TYPE

Boolean

DEFAULT

false

GROUP

power

DESCRIPTION

Dynamic power optimization during **compile** can be invoked by **set_max_dynamic_power** or **set_max_total_power** command. This variable causes **compile** command to invoke extra dyanmic power optimization algorithms to further reduce the dynamic power. Since those algorithms are runtime intensive, they are not turned on by default. Be aware of the runtime overhead when using this variable.

The following example will cause the **compile -incr** to run extra algorithms for better dynamic power.

```
   prompt> set power_opto_extra_high_dynamic_power_effort true
   prompt> set_max_dynamic_power 0.0
   prompt> compile -incr
```

SEE ALSO

compile(2)
set_max_dynamic_power(2)
set_max_total_power(2)

power_preserve_rtl_hier_names

Preserves the hierarchy information of the RTL objects in the RTL design.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

When *true*, HDL Compiler preserves the hierarchy information of the RTL objects in the RTL design. The **rtl2saif** command, which generates RTL forward-annotation SAIF files, needs this information. When this variable is *false* (the default value), **rtl2saif** cannot extract the correct synthesis invariant objects because the hierarchy information is not preserved.

To determine the current value of this variable, type **printvar power_preserve_rtl_hier_names**. For a list of power variables and their current values, type **print_variable_group power**.

SEE ALSO

rtl2saif(2)

power_rclock_inputs_use_clocks_fanout

Specifies whether clock network objects in an input port fanout are used to infer the input port's related clock.

TYPE

Boolean

DEFAULT

true

GROUP

power_variables

DESCRIPTION

This variable is used during related clock inference to decide whether the inferred related clock on design port is chosen to be the fastest clock whose clock network objects are in the port's transitive fanout. For example, if this variable is set to **true**, then if the transitive fanout of an input port contains a number of cells, then the fastest clock on these flip-flop cells is chosen as the inferred related clock on the input port. If the variable is set to **false**, then the input port will not have an inferred related clock.

For more information on the mechanism used to infer related clock information, please refer to the man page of the **propagate_switching_activity** command.

SEE ALSO

propagate_switching_activity(2)
power_rclock_use_asynch_inputs(3)
power_rclock_unrelated_use_fastest(3)

power_rclock_unrelated_use_fastest

Specifies whether the fastest clock is set as the related clock of a design object when a related clock is not inferred by the related clock inference mechanism.

TYPE

Boolean

DEFAULT

true

GROUP

power_variables

DESCRIPTION

This variable is used during the last stage of related clock inference to decide whether the design objects that do not have an inferred related clock will be set a related clock or not. The user can use the **set_switching_activity** command with the argument **-clock "*"** to specify that Power Compiler will automatically infer the related clocks for any specified objects. If the related clock inference mechanism did not infer a related clock for a number of such objects, then, when the value of the **power_rclock_unrelated_use_fastest** is true, Power Compiler will set the fastest design clock as the objects' related clock. When the variable is set to **false**, such objects will not have a related clock.

For more information on the mechanism used to infer related clock information, please refer to the man page of the **propagate_switching_activity** command.

SEE ALSO

propagate_switching_activity(2)
power_rclock_inputs_use_clocks_fanout(3)
power_rclock_use_asynch_inputs(3)

power_rclock_use_asynch_inputs

Specifies whether the inferred related clock on an asynchronous pin of a flip-flop is used to determine the nferred related clock on the cell's outputs.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

This variable is used during related clock inference to decide whether the inferred related clock a flip-flop cell output considers the inferred related clocks on the cell's asynchronous inputs. When this variable is set to **false**, then the inferred related clock on a flip-flop cell output is the inferred related clock on the cell's clock pin. When this variable is set to **true**, then the inferred related clock on a flip-flop cell output is the fastest inferred clock on the cell's clock pin and asynchronous input pins.

For more information on the mechanism used to infer related clock information, please refer to the man page of the **propagate_switching_activity** command.

SEE ALSO

propagate_switching_activity(2)
power_rclock_inputs_use_clocks_fanout(3)
power_rclock_use_asynch_inputs(3)

power_remove_redundant_clock_gates

Specifies to the **compile -incremental** and **physopt -incremental commands whether to remove redundant Synopsys clock gating cells.**

TYPE

Boolean

DEFAULT

true

GROUP

power_variables

DESCRIPTION

Specifies whether to remove redundant Synopsys clock gating cells during incremental compile. A clock gate is considered redundant when it is always enabled. This is the case when the enable net is tied to logic one. If this variable is set to *true* (the default value), the redundant clock-gating cells are removed during incremental compile. To disable automatic removal of redundant clock gating cells, set the value to *false* before issuing the **compile** or **physopt** commands.

This variable is valid only with Design Compiler (dc_shell, dc_shell-t) and Physical Compiler (psyn_shell).

To determine the current value of this variable, type **printvar power_remove_redundant_clock_gates**. For a list of power variables and their current values, type **print_variable_group power**.

SEE ALSO

compile(2)
power_variables(3)

power_rtl_saif_file

Defines for the **rtl2saif** command where to store the forward-annotation SAIF file, if you do not specify the -output option.

TYPE

Boolean

DEFAULT

power_rtl.saif

GROUP

power_variables

DESCRIPTION

Defines for the **rtl2saif** command where to store the forward-annotation SAIF file, if you do not specify the -output option. The default for this variable is *power_rtl.saif*.

This variable is valid only under dc_shell (Design Compiler).

To determine the current value of this variable, type **printvar power_rtl_saif_file**. For a list of power variables and their current values, type **print_variable_group power**.

SEE ALSO

rtl2saif(2)

power_sa_propagation_effort

Specifies the default effort level used when propagating switching activity.

TYPE

string

DEFAULT

low

GROUP

power_variables

DESCRIPTION

Power calculations need switching activity information on all design nets, and Power Compiler uses a switching activity propagation mechanism to estimate the activity on the nets that are not annotated by the user. This variable specifies the default effort level used by the switching activity propagation mechanism.

The following Power Compiler commands may use the propagation mechanism: **report_power**, synthesis commands like **compile** when used with power constraints, and **write_saif** when specified with the **-propagated** option. The **propagate_switching_activity** command can be used to force a re-propagation of the switching activity.

The **-analysis_effort** argument of the **report_power** command, and the **-effort** argument of the **propagate_switching_activity** command, specify the effort level used by the switching activity propagation mechanism. When these options are not specified, the value of the **power_sa_propagation_effort** variable is used.

The **power_sa_propagation_effort** variable can have the following values: **low**, **medium**, **high**.

SEE ALSO

propagate_switching_activity(2)
power_sa_propagation_verbose(3)

power_sa_propagation_verbose

Specifies the default verbose mode used when propagating switching activity.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

Power calculations need switching activity information on all design nets, and Power Compiler uses a switching activity propagation mechanism to estimate the activity on the nets that are not annotated by the user. This variable specifies the default verbose level used when propagating switching activity. During verbose mode, the propagation mechanism displays more warning and information messages.

The following Power Compiler commands may use the propagation mechanism: **report_power**, synthesis commands like **compile** when used with power constraints, and **write_saif** when specified with the **-propagated** option. The **propagate_switching_activity** command can be used to force a re-propagation of the switching activity.

The **-verbose** flag of the **propagate_switching_activity** command can be used to propagate the switching activity in verbose mode.

SEE ALSO

propagate_switching_activity(2)
power_sa_propagation_effort(3)

power_same_switching_activity_on_connected_objects

Forces the tool to use the last user-annotated switching activity data on all connected tool objects.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

The **power_same_switching_activity_on_connected_objects** variable forces all switching activity data of all connected tool objects to use the latest user-annotated switching activity data. The user-annotated switching activity information can come from the **read_saif** command or the **set_switching_activity** command. Power reporting commands may produce different power results, since the switching activity data may be changed.

The **power_same_switching_activity_on_connected_objects** variable is off (false) by default.

SEE ALSO

read_saif(2)
set_switching_activity(2)

power_sdpd_message_tolerance

Specifies the tolerance value for issuing warnings and information messages during fixing of user-annotated SDPD switching activity.

TYPE

Float

DEFAULT

0.00001

GROUP

power_variables

DESCRIPTION

This variable specifies a tolerance value that is used when messages are reported during the SDPD fixing step. The SDPD fixing step is enabled by the **power_fix_sdpd_annotation** variable; please check the man page of **power_fix_sdpd_annotation** for more information on this step. Messages reported by the SDPD fixing step need to specify the tolerance criteria specifid by this variable. For example, SDPD fixing scales state-dependent static probabilities so that they add up to 1.0. A message is reported by the SDPD fixing step if the difference between the sum of the state-dependent static probabilites and 1.0 is not within the tolerance value. A tolerance value of 0.0 makes the SDPD fixing step report a message for every check/modification to the SDPD annotation, however this will most likely report non-issues due to floating point errors. A high value of the **power_sdpd_message_tolerance** value will filter out all but the most significant messages. Also note, that the verbosity of the SDPD fixing step is determined by the **power_fix_sdpd_annotation_verbose** variable. When **power_fix_sdpd_annotation_verbose** is set to false most of the messages reported by the SDPD fixing step are replaced by a single message indicating that user-annotated SDPD switching activity is being corrected.

SEE ALSO

power_fix_sdpd_annotation(3)
power_fix_sdpd_annotation_verbose(3)

power_sdpd_saif_file

Defines for the **lib2saif** command where to store the forward-annotation SAIF file, if you do not specify the -output option.

TYPE

Boolean

DEFAULT

power_sdpd.saif

GROUP

power_variables

DESCRIPTION

Defines for the **lib2saif** command where to store the forward-annotation SAIF file, if you do not specify the -output option. The default for this variable is *power_sdpd.saif*.

This variable is valid only under dc_shell (Design Compiler).

To determine the current value of this variable, type **printvar power_sdpd_saif_file**. For a list of power variables and their current values, type **print_variable_group power**.

SEE ALSO

lib2saif(2)

power_variables

Variables that affect Power Compiler commands.

SYNTAX

float   **power_default_static_probability** = "0.5"
float   **power_default_toggle_rate** = "0.1"
Boolean **power_do_not_size_icg_cells** = "false"
Boolean **power_hdlc_do_not_split_cg_cells** = "false"
Boolean **power_keep_license_after_power_commands** = "true"
Boolean **power_preserve_rtl_hier_names** = "false"
string **power_rtl_saif_file** = "power_rtl.saif"
string **power_sa_propagation_effort** = "low"
Boolean **power_sa_propagation_verbose** = "false"
string **power_sdpd_saif_file** = "power_sdpd.saif"
Boolean **power_remove_redundant_clock_gates** = "true"
Boolean **power_driven_clock_gating** = "false"
Boolean **power_cg_all_registers** = "false"

Boolean **power_cg_derive_related_clock** = "false"
Boolean **power_cg_ignore_setup_condition** = "false"
Boolean **power_cg_print_enable_conditions** = "false"
int   **power_cg_print_enable_conditions_max_terms** = "10"
Boolean **do_operand_isolation** = "true"
Boolean **power_cg_enable_alternative_algorithm** = "false"

DESCRIPTION

These variables directly affect the Power Compiler commands. Default values are shown above in the Syntax section.

For a list of **power** variables and their current values, type **print_variable_group power**. To view this manual page online, type **man power_variables**. To view an individual variable description, type **man *var***, where *var* is the name of the variable.

power_default_static_probability
: The static probability value on unannotated primary input nets and black box outputs used during power calculations. The value of this variable must be between 0.0 and 1.0, otherwise it will be ignored and the default value of *0.5* is used.

power_default_toggle_rate
: This variable specifies the toggle rate value used for unannotated nets driven by primary inputs and black box outputs. For such nets, power calculations assume a toggle rate value derived by multiplying the value of **power_default_toggle_rate** with the frequency of the fastest clock in the design. The value of this variable must be a non-negative number, otherwise the default value of *0.1* is used.

power_do_not_size_icg_cells
: When *false* (the default), the **compile** command sizes integrated clock-gating

cells in the design to correct DRC violations. When *true*, **compile** does not size the cells. Set this variable to *true* if your clock tree synthesis (CTS) tool inserts buffers after the integrated cell, and you do not want to fix DRC violations before the CTS buffers are inserted.

power_hdlc_do_not_split_cg_cells

When *false* (the default), **elaborate** splits clock-gating cells to limit their fanout, controlled by the value specified by the **set_clock_gating_style -max_fanout** command, whose default is 128. If *true*, **elaborate** does not split clock-gating cells, resulting in a netlist where all registers are gated by a single clock-gating cell if they share the same enable signal. **elaborate** does not honor the **set_clock_gating_style -max_fanout** value specified. Set this variable to *true* if your clock tree synthesis (CTS) tool inserts buffers after the clock-gating cell, and you do not want **elaborate** to limit the fanout of the clock-gating cell.

power_keep_license_after_power_commands

When *true*, a Power Compiler license that is checked out under **dc_shell** (DesignCompiler) will remain checked out throughout the **dc_shell** session. When this variable is *false* (the default value), the Power Compiler license will remain checked out only as long as a command is using it. At the completion of the command, the license will be released.

power_preserve_rtl_hier_names

When *true*, HDL compiler will preserve the hierarchy information of the RTL objects in the RTL design. The information is needed by *rtl2saif* which generates RTL forward annotation SAIF files. When this variable is *false* (the default value), *rtl2saif* will not be able to extract the correct synthesis invariant objects since the hierarchy information is not preserved.

power_rtl_saif_file

This is the name of the file where *rtl2saif* will store the forward annotation saif if the **-output** option is not specified for this command.

power_sa_propagation_effort

The default effort level used during switching activity propagation. Valid values are: *low* (the default), *medium* and *high*.

power_sa_propagation_verbose

The default verbose level used during switching activity propagation. During verbose mode the switching activity propagation mechanism displays more warning messages, for example, when the design contains primary input nets and black-box outputs without switching activity annotation.

power_sdpd_saif_file

This is the name of the file where *lib2saif* will store the forward annotation saif if the -output option is not specified for this command.

power_remove_redundant_clock_gates

When *true*, redundant Synopsys clock gating cells will be automatically removed during incremental compile.

power_driven_clock_gating

Controls whether switching activity and dynamic power of the register banks should be considered when optimizing the clock gating of the design.

power_cg_all_registers
    When *true*, a (potentially redundant) clock gate will be inserted for all registers.

power_cg_derive_related_clock
    When *true*, clock domain relationship between registers will be derived from the hierarchical context.

power_cg_ignore_setup_condition
    When *true*, the setup condition will be ignored for latch-free clock gating.

power_cg_print_enable_conditions
    When *true*, the enable conditions of registers and clock gates will be reported during clock gate insertion.

power_cg_print_enable_conditions_max_terms
    Specifies the maximum number of product terms to be reported in the sum of product expansion of the enable condition.

power_cg_enable_alternative_algorithm
    When *true*, an algorithm different from the default algorithm will be used to find gatable registers. For larger designs it may shorten the runtime for clock gating.

do_operand_isolation
    When *true*, enables operand isolation as a dynamic power optimization technique for a design.

SEE ALSO

report_power (2), **lib2saif** (2), **rtl2saif** (2), **set_clock_gating_style** (2), **insert_clock_gating** (2), **elaborate** (2), **compile** (2); **power_default_static_probability** (3), **power_default_toggle_rate** (3), **power_do_not_size_icg_cells** (3), **power_hdlc_do_not_split_cg_cells** (3), **power_keep_license_after_power_commands** (3), **power_rtl_saif_file** (3), **power_sa_propagation_effort** (3), **power_sa_propagation_verbose** (3), **power_sdpd_saif_file** (3), **power_remove_redundant_clock_gates** (3), **power_cg_enable_alternative_algorithm** (3), **do_operand_isolation** (3).

preview_scan_variables

SYNTAX

Boolean **test_jump_over_bufs_invs** = "1"
string **test_preview_scan_shows_cell_types** = "FALSE"
string **test_scan_link_so_lockup_key** = "l"
string **test_scan_link_wire_key** = "w"
string **test_scan_segment_key** = "s"
string **test_scan_true_key** = "t"

DESCRIPTION

These variables directly affect the **preview_scan** command. Defaults are shown above, under Syntax.

For a list of **preview_scan** variables, type **print_variable_group preview_scan**. To view this manual page on-line, type **help preview_scan_variables**. To view an individual variable description, type **help *variable-name***, where *variable-name* is the name of the variable.

test_jump_over_bufs_invs
    Determines whether the inernal clock detection mechanism considers the output pins of buffers and inverters as internal clocks. The default is *true*.

test_preview_scan_shows_cell_types
    Specifies whether **preview_scan** shows cell types.

test_scan_link_so_lockup_key
    Specifies the key **preview_scan** uses to report cells whose scan-outs drive lockup latches.

test_scan_link_wire_key
    Specifies the key **preview_scan** uses to report cells whose scan-outs drive wire scan links.

test_scan_segment_key
    Specifies the key **preview_scan** uses to report scan segments.

test_scan_true_key
    Specifies the key **preview_scan** uses to report cells with true scan attributes.

SEE ALSO

preview_scan (2), **test_jump_over_bufs_invs** (3), **test_preview_scan_shows_cell_types** (3), **test_scan_link_so_lockup_key** (3), **test_scan_link_wire_key** (3), **test_scan_segment_key** (3), **test_scan_true_key** (3).

psyn_stress_map

Enables generation of stress map. Set this variable to true to generate stress maps from Physical Compiler.

TYPE

Boolean

DEFAULT

false

GROUP

physopt

DESCRIPTION

Enables generation of stress map. The stress map shows how intensively the optimization (timing, DRC, power and others) methods were applied on the current design. Set this variable to true to generate stress maps from Physical Compiler. The default value is false. It must be set to true before the physopt command to generate stress maps.

To determine the current value of this variable, enter the following command:

```
psyn_shell-t> printvar psyn_stress_map
```

rc_degrade_min_slew_when_rd_less_than_rnet

Enables or disables the use of slew degradation in min analysis mode during the RCCALC-009 condition.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When *false*,(the default), slew degradation through RC networks is not used in min analysis mode during the RCCALC-009 condition. When *true*, slew degradation is used during the RCCALC-009 condition.

The "RCCALC-009 condition" means a condition in which timing analysis checks the library-derived drive resistance, and if it is less than the dynamic RC network impedance to ground by an amount equal to or greater than the value of a particular drive-strength threshold, timing analysis adjusts the drive resistance using an empirical formula to improve accuracy, and issues the RCCALC-009 message. In case this improved accuracy is not sufficient, timing analysis provides extra pessimism by not using slew degradation in min analysis mode; however, superfluous min delay violations could occur as a side effect. You can keep slew degradation on in min analysis mode after you have qualified the RCCALC-009 methodology for your accuracy requirements, by setting this variable to *true*.

To determine the current value of this variable, type **printvar rc_degrade_min_slew_when_rd_less_than_rnet** or **echo $rc_degrade_min_slew_when_rd_less_than_rnet**.

SEE ALSO

rc_driver_model_mode

Specifies which driver model type to use for RC delay calculation.

TYPE

String

DEFAULT

basic

GROUP

timing_variables

DESCRIPTION

DesignTime supports two types of driver models for RC delay calculation, basic and advanced. The basic model is derived from the conventional delay and slew library schema, while the advanced model is derived from a new schema. The advanced driver model is part of the Synopsys Composite Current-Source (CCS) model.

When the shell variable rc_driver_model_mode is set to basic, RC delay calculation will always use driver models derived from the conventional delay and slew schema present in design libraries. When set to advanced, RC delay calculation will use the advanced driver model if data for it is present. The report_delay_calculation command used on a cell arc will show the message "Advanced driver-modeling used" as appropriate.

When the shell variable rc_driver_model_mode is set to basic, and the variable rc_receiver_model_mode is set to advanced, DesignTime will use the advanced voltage-dependent capacitance models to derive an equivalent single capacitance dependent only on the rise, fall, min or max arc condition and these equivalent capacitances will be used in analysis instead of the pin capacitances from the library. Please check the manpage for variable rc_receiver_model_mode for additional details.

To determine the current value of this variable, enter the following command:

```
prompt> printvar rc_driver_model_mode
```

SEE ALSO

rc_receiver_model_mode(3)
report_delay_calculation(2)

rc_input_threshold_pct_fall

Specifies the threshold voltage that defines the startpoint of the falling cell or net delay calculation.

TYPE

float

DEFAULT

50.000000

GROUP

timing_variables

DESCRIPTION

Specifies the threshold voltage that defines the startpoint of the falling cell or net delay calculation. The value is a percent of the voltage source. Allowed values are 0.0 - 100.0 inclusive; the default is 50.0.

This variable is one of 8 variables, listed in Table 1, that affect delay and transition time computations for detailed RC networks. These variables interpret the cell delays and transition times from the Synopsys library. The values specified by these variables are overridden by trippoint values specified in the library so normally it should not be necessary to set the variables. The values specified are only applied to libraries that do not contain trippoint specifications.

Table 1

Variable Name	**Default**
rc_slew_lower_threshold_pct_rise	**20.0**
rc_slew_lower_threshold_pct_fall	**20.0**
rc_slew_upper_threshold_pct_rise	**80.0**
rc_slew_upper_threshold_pct_fall	**80.0**
rc_input_threshold_pct_rise	**50.0**
rc_input_threshold_pct_fall	**50.0**
rc_output_threshold_pct_rise	**50.0**
rc_output_threshold_pct_fall	**50.0**

The default values specify that a cell delay is defined from 50% of the voltage value for the input transition to 50% of the voltage value for the output transition. The default values also specify that a transition time, or slew, is defined from 20% to 80% of the voltage.

To determine the current value of this variable, type **printvar rc_input_threshold_pct_fall**.

For a list of all timing variables and their current values, type **print_variable_group timing**.

EXAMPLES

The following example specifies that cell delays from the Synopsys library are computed from 50% of the input transition to 55% of the output transition. In addition, the example specifies that transition times in the Synopsys library represent the delay from 10% to 90% of the voltage source.

psyn_shell-t> **set rc_slew_lower_threshold_pct_fall 10** psyn_shell-t> **set rc_slew_lower_threshold_pct_rise 10** psyn_shell-t> **set rc_slew_upper_threshold_pct_fall 90** psyn_shell-t> **set rc_slew_upper_threshold_pct_rise 90** psyn_shell-t> **set rc_input_threshold_pct_rise 50** psyn_shell-t> **set rc_input_threshold_pct_fall 50** psyn_shell-t> **set rc_output_threshold_pct_rise 55** psyn_shell-t> **set rc_output_threshold_pct_fall 55**

SEE ALSO

rc_input_threshold_pct_rise(3)
rc_input_threshold_pct_fall(3)
rc_output_threshold_pct_fall(3)
rc_output_threshold_pct_rise(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_rise(3)
lib_thresholds_per_lib(3)

rc_input_threshold_pct_rise

Specifies the threshold voltage that defines the startpoint of the rising cell or net delay calculation.

TYPE

float

DEFAULT

50.000000

GROUP

timing_variables

DESCRIPTION

Specifies the threshold voltage that defines the startpoint of the rising cell or net delay calculation. The value is a percent of the voltage source. Allowed values are 0.0 - 100.0 inclusive; the default is 50.0.

This variable is one of 8 variables, listed in Table 1, that affect delay and transition time computations for detailed RC networks. These variables interpret the cell delays and transition times from the Synopsys library. The values specified by these variables are overridden by trippoint values specified in the library so normally it should not be necessary to set the variables. The values specified are only applied to libraries that do not contain trippoint specifications.

Table 1

Variable Name	**Default**
rc_slew_lower_threshold_pct_rise	**20.0**
rc_slew_lower_threshold_pct_fall	**20.0**
rc_slew_upper_threshold_pct_rise	**80.0**
rc_slew_upper_threshold_pct_fall	**80.0**
rc_input_threshold_pct_rise	**50.0**
rc_input_threshold_pct_fall	**50.0**
rc_output_threshold_pct_rise	**50.0**
rc_output_threshold_pct_fall	**50.0**

The default values specify that a cell delay is defined from 50% of the voltage value for the input transition to 50% of the voltage value for the output transition. The default values also specify that a transition time, or slew, is defined from 20% to 80% of the voltage.

To determine the current value of this variable, type **printvar rc_input_threshold_pct_rise**.

For a list of all timing variables and their current values, type **print_variable_group timing**.

EXAMPLES

The following example specifies that cell delays from the Synopsys library are computed from 50% of the input transition to 55% of the output transition. In addition, the example specifies that transition times in the Synopsys library represent the delay from 10% to 90% of the voltage source.

```
psyn_shell-t> set rc_slew_lower_threshold_pct_fall 10 psyn_shell-t> set
rc_slew_lower_threshold_pct_rise 10 psyn_shell-t> set
rc_slew_upper_threshold_pct_fall 90 psyn_shell-t> set
rc_slew_upper_threshold_pct_rise 90 psyn_shell-t> set rc_input_threshold_pct_rise 50
psyn_shell-t> set rc_input_threshold_pct_fall 50 psyn_shell-t> set
rc_output_threshold_pct_rise 55 psyn_shell-t> set rc_output_threshold_pct_fall 55
```

SEE ALSO

rc_input_threshold_pct_rise(3)
rc_input_threshold_pct_fall(3)
rc_output_threshold_pct_fall(3)
rc_output_threshold_pct_rise(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_rise(3)
lib_thresholds_per_lib(3)

rc_noise_model_mode

When set to **advanced**, enables the use of CCS noise, if available in the design library.

TYPE

string

DEFAULT

basic

GROUP

signal integrity

DESCRIPTION

When set to **advanced**, enables CCS noise in static noise analysis and optimization.

When basic (the default), CCS noise information is not used even if it exists in the library. However, if the library has NLDM noise information, it is still used.

To determine the current value of this variable, use **printvar rc_noise_model_mode**.

SEE ALSO

rc_output_threshold_pct_fall

Specifies the threshold voltage that defines the startpoint of the falling cell or net delay calculation.

TYPE

float

DEFAULT

50.000000

GROUP

timing_variables

DESCRIPTION

Specifies the threshold voltage that defines the startpoint of the falling cell or net delay calculation. The value is a percent of the voltage source. Allowed values are 0.0 - 100.0 inclusive; the default is 50.0.

This variable is one of 8 variables, listed in Table 1, that affect delay and transition time computations for detailed RC networks. These variables interpret the cell delays and transition times from the Synopsys library. The values specified by these variables are overridden by trippoint values specified in the library so normally it should not be necessary to set the variables. The values specified are only applied to libraries that do not contain trippoint specifications.

Table 1

Variable Name	**Default**
rc_slew_lower_threshold_pct_rise	**20.0**
rc_slew_lower_threshold_pct_fall	**20.0**
rc_slew_upper_threshold_pct_rise	**80.0**
rc_slew_upper_threshold_pct_fall	**80.0**
rc_input_threshold_pct_rise	**50.0**
rc_input_threshold_pct_fall	**50.0**
rc_output_threshold_pct_rise	**50.0**
rc_output_threshold_pct_fall	**50.0**

The default values specify that a cell delay is defined from 50% of the voltage value for the input transition to 50% of the voltage value for the output transition. The default values also specify that a transition time, or slew, is defined from 20% to 80% of the voltage.

To determine the current value of this variable, type **printvar rc_output_threshold_pct_fall**.

For a list of all timing variables and their current values, type **print_variable_group timing**.

EXAMPLES

The following example specifies that cell delays from the Synopsys library are computed from 50% of the input transition to 55% of the output transition. In addition, the example specifies that transition times in the Synopsys library represent the delay from 10% to 90% of the voltage source.

psyn_shell-t> **set rc_slew_lower_threshold_pct_fall 10** psyn_shell-t> **set rc_slew_lower_threshold_pct_rise 10** psyn_shell-t> **set rc_slew_upper_threshold_pct_fall 90** psyn_shell-t> **set rc_slew_upper_threshold_pct_rise 90** psyn_shell-t> **set rc_input_threshold_pct_rise 50** psyn_shell-t> **set rc_input_threshold_pct_fall 50** psyn_shell-t> **set rc_output_threshold_pct_rise 55** psyn_shell-t> **set rc_output_threshold_pct_fall 55**

SEE ALSO

rc_input_threshold_pct_rise(3)
rc_input_threshold_pct_fall(3)
rc_output_threshold_pct_fall(3)
rc_output_threshold_pct_rise(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_rise(3)
lib_thresholds_per_lib(3)

rc_output_threshold_pct_rise

Specifies the threshold voltage that defines the startpoint of the rising cell or net delay calculation.

TYPE

float

DEFAULT

50.000000

GROUP

timing_variables

DESCRIPTION

Specifies the threshold voltage that defines the startpoint of the rising cell or net delay calculation. The value is a percent of the voltage source. Allowed values are 0.0 - 100.0 inclusive; the default is 50.0.

This variable is one of 8 variables, listed in Table 1, that affect delay and transition time computations for detailed RC networks. These variables interpret the cell delays and transition times from the Synopsys library. The values specified by these variables are overridden by trippoint values specified in the library so normally it should not be necessary to set the variables. The values specified are only applied to libraries that do not contain trippoint specifications.

Table 1

Variable Name	Default
rc_slew_lower_threshold_pct_rise	**20.0**
rc_slew_lower_threshold_pct_fall	**20.0**
rc_slew_upper_threshold_pct_rise	**80.0**
rc_slew_upper_threshold_pct_fall	**80.0**
rc_input_threshold_pct_rise	**50.0**
rc_input_threshold_pct_fall	**50.0**
rc_output_threshold_pct_rise	**50.0**
rc_output_threshold_pct_fall	**50.0**

The default values specify that a cell delay is defined from 50% of the voltage value for the input transition to 50% of the voltage value for the output transition. The default values also specify that a transition time, or slew, is defined from 20% to 80% of the voltage.

To determine the current value of this variable, type **printvar rc_output_threshold_pct_rise**.

For a list of all timing variables and their current values, type **print_variable_group timing**.

EXAMPLES

The following example specifies that cell delays from the Synopsys library are computed from 50% of the input transition to 55% of the output transition. In addition, the example specifies that transition times in the Synopsys library represent the delay from 10% to 90% of the voltage source.

psyn_shell-t> **set rc_slew_lower_threshold_pct_fall 10** psyn_shell-t> **set rc_slew_lower_threshold_pct_rise 10** psyn_shell-t> **set rc_slew_upper_threshold_pct_fall 90** psyn_shell-t> **set rc_slew_upper_threshold_pct_rise 90** psyn_shell-t> **set rc_input_threshold_pct_rise 50** psyn_shell-t> **set rc_input_threshold_pct_fall 50** psyn_shell-t> **set rc_output_threshold_pct_rise 55** psyn_shell-t> **set rc_output_threshold_pct_fall 55**

SEE ALSO

rc_input_threshold_pct_rise(3)
rc_input_threshold_pct_fall(3)
rc_output_threshold_pct_fall(3)
rc_output_threshold_pct_rise(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_rise(3)
lib_thresholds_per_lib(3)

rc_receiver_model_mode

Specifies which receiver model type to use for RC delay calculation.

TYPE

String

DEFAULT

basic

GROUP

timing_variables

DESCRIPTION

DesignTime supports two types of receiver models for RC delay calculation, basic and advanced. The basic model is a single capacitance dependent only on the rise, fall, min, or max arc condition. The advanced model is a voltage-dependent capacitance additionally dependent on input-slew and output capacitance. The advanced model has many advantages, one of which is that the accuracy of both delays and slews is improved. Another advantage is that nonlinearities such as the Miller effect are addressed. The advanced receiver model is part of the Synopsys Composite Current-Source (CCS) model.

When set to advanced, RC delay calculation will use the advanced receiver model if data for it is present and if the network is driven by the advanced driver model. The report_delay_calculation command used on a network arc will show the message "Advanced receiver-modeling used" as appropriate.

When the shell variable rc_receiver_model_mode is set to advanced, and the network is not driven by the advanced driver model, (i.e. the variable rc_driver_model_mode is set to basic or lumped load is used), DesignTime will use the advanced voltage-dependent capacitance models to derive an equivalent single capacitance dependent only on the rise, fall, min or max arc condition. These equivalent capacitances will be used in analysis instead of the pin capacitances from the library. The report_delay_calculation command used on a network arc will not show the message "Advanced receiver-modeling used" for these calculations, since only an equivalent single capacitance is used.

When the shell variable rc_receiver_model_mode is set to basic, RC delay calculation will always use the pin capacitances specified in the design libraries.

To determine the current value of this variable, enter the following command:

```
prompt> printvar rc_receiver_model_mode
```

SEE ALSO

```
rc_driver_model_mode(3)
report_delay_calculation(2)
```

rc_slew_derate_from_library

Specifies the derating needed for the transition times in the Synopsys library to match the transition times between the characterization trip points.

TYPE

float

DEFAULT

1.000000

GROUP

timing_variables

DESCRIPTION

A floating point number between 0.0 and 1.0 that specifies the derating needed for the transition times in the Synopsys library to match the transition times between the characterization trip points. The default is 1.0, which means that the transition times in the Synopsys library are used without change.

The value this variable specifies is overridden by any library-specified slew-derating values, so normally it should not be necessary to set the variable. The value specified applies only to libraries that do not contain slew-derating specifications.

A slew-derating value of 1.0 should be used if the transition times specified in the library represent the exact transition times between the characterization trip points, which is usually the case. Use a slew-derating value of less than 1.0 for libraries where the transition times have been extrapolated to the rail voltages. For example, if the transition times are characterized as between 30% and 70% and then extrapolated to the rails, the slew-derating value is 0.4 = (70 - 30) / 100.

To determine the current value of this variable, type **printvar rc_slew_derate_from_library**.

For a list of all timing variables and their current values, type **print_variable_group timing**.

EXAMPLES

The following example specifies that cell delays from the Synopsys library have been computed from 50 percent of the input transition to 50 percent of the output transition. The example also specifies that transition times in the Synopsys library were computed by measuring the delay from 30 percent to 70 percent of the voltage source and then multiplying the measured transition times by 2.5 = (100-0)/(70-30) to extrapolate to 0-100 percent of the rail voltages.

```
psyn_shell-t> set rc_slew_derate_from_library 0.4
psyn_shell-t> set rc_slew_lower_threshold_pct_fall 30
psyn_shell-t> set rc_slew_lower_threshold_pct_rise 30
psyn_shell-t> set rc_slew_upper_threshold_pct_fall 70
psyn_shell-t> set rc_slew_upper_threshold_pct_rise 70
psyn_shell-t> set rc_input_threshold_pct_rise      50
psyn_shell-t> set rc_input_threshold_pct_fall      50
psyn_shell-t> set rc_output_threshold_pct_rise     55
psyn_shell-t> set rc_output_threshold_pct_fall     55
```

SEE ALSO

```
lib_thresholds_per_lib(3)
rc_input_threshold_pct_rise(3)
rc_input_threshold_pct_fall(3)
rc_output_threshold_pct_fall(3)
rc_output_threshold_pct_rise(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_rise(3)
```

rc_slew_lower_threshold_pct_fall

Specifies the threshold voltage that defines the endpoint of the falling slew calculation.

TYPE

float

DEFAULT

20.000000

GROUP

timing_variables

DESCRIPTION

Specifies the threshold voltage that defines the endpoint of the falling slew calculation. The value is a percent of the voltage source. Allowed values are 0.0 - 100.0 inclusive; the default is 20.0.

This variable is one of 8 variables, listed in Table 1, that affect delay and transition time computations for detailed RC networks. These variables interpret the cell delays and transition times from the Synopsys library. The values specified by these variables are overridden by trippoint values specified in the library so normally it should not be necessary to set the variables. The values specified are only applied to libraries that do not contain trippoint specifications.

Table 1

Variable Name	**Default**
rc_slew_lower_threshold_pct_rise	**20.0**
rc_slew_lower_threshold_pct_fall	**20.0**
rc_slew_upper_threshold_pct_rise	**80.0**
rc_slew_upper_threshold_pct_fall	**80.0**
rc_input_threshold_pct_rise	**50.0**
rc_input_threshold_pct_fall	**50.0**
rc_output_threshold_pct_rise	**50.0**
rc_output_threshold_pct_fall	**50.0**

The default values specify that a cell delay is defined from 50% of the voltage value for the input transition to 50% of the voltage value for the output transition. The default values also specify that a transition time, or slew, is defined from 20% to 80% of the voltage.

To determine the current value of this variable, type **printvar rc_slew_lower_threshold_pct_fall**.

For a list of all timing variables and their current values, type **print_variable_group timing**.

EXAMPLES

The following example specifies that cell delays from the Synopsys library are computed from 50% of the input transition to 55% of the output transition. In addition, the example specifies that transition times in the Synopsys library represent the delay from 10% to 90% of the voltage source.

psyn_shell-t> **set rc_slew_lower_threshold_pct_fall 10** psyn_shell-t> **set rc_slew_lower_threshold_pct_rise 10** psyn_shell-t> **set rc_slew_upper_threshold_pct_fall 90** psyn_shell-t> **set rc_slew_upper_threshold_pct_rise 90** psyn_shell-t> **set rc_input_threshold_pct_rise 50** psyn_shell-t> **set rc_input_threshold_pct_fall 50** psyn_shell-t> **set rc_output_threshold_pct_rise 55** psyn_shell-t> **set rc_output_threshold_pct_fall 55**

SEE ALSO

rc_input_threshold_pct_rise(3)
rc_input_threshold_pct_fall(3)
rc_output_threshold_pct_fall(3)
rc_output_threshold_pct_rise(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_rise(3)
lib_thresholds_per_lib(3)

rc_slew_lower_threshold_pct_rise

Specifies the threshold voltage that defines the startpoint of the rising slew calculation.

TYPE

float

DEFAULT

20.000000

GROUP

timing_variables

DESCRIPTION

Specifies the threshold voltage that defines the startpoint of the rising slew calculation. The value is a percent of the voltage source. Allowed values are 0.0 - 100.0 inclusive; the default is 20.0.

This variable is one of 8 variables, listed in Table 1, that affect delay and transition time computations for detailed RC networks. These variables interpret the cell delays and transition times from the Synopsys library. The values specified by these variables are overridden by trippoint values specified in the library so normally it should not be necessary to set the variables. The values specified are only applied to libraries that do not contain trippoint specifications.

Table 1

Variable Name	**Default**
rc_slew_lower_threshold_pct_rise	**20.0**
rc_slew_lower_threshold_pct_fall	**20.0**
rc_slew_upper_threshold_pct_rise	**80.0**
rc_slew_upper_threshold_pct_fall	**80.0**
rc_input_threshold_pct_rise	**50.0**
rc_input_threshold_pct_fall	**50.0**
rc_output_threshold_pct_rise	**50.0**
rc_output_threshold_pct_fall	**50.0**

The default values specify that a cell delay is defined from 50% of the voltage value for the input transition to 50% of the voltage value for the output transition. The default values also specify that a transition time, or slew, is defined from 20% to 80% of the voltage.

To determine the current value of this variable, type **printvar rc_slew_lower_threshold_pct_rise**.

For a list of all timing variables and their current values, type **print_variable_group timing**.

EXAMPLES

The following example specifies that cell delays from the Synopsys library are computed from 50% of the input transition to 55% of the output transition. In addition, the example specifies that transition times in the Synopsys library represent the delay from 10% to 90% of the voltage source.

```
psyn_shell-t> set rc_slew_lower_threshold_pct_fall 10 psyn_shell-t> set
rc_slew_lower_threshold_pct_rise 10 psyn_shell-t> set
rc_slew_upper_threshold_pct_fall 90 psyn_shell-t> set
rc_slew_upper_threshold_pct_rise 90 psyn_shell-t> set rc_input_threshold_pct_rise 50
psyn_shell-t> set rc_input_threshold_pct_fall 50 psyn_shell-t> set
rc_output_threshold_pct_rise 55 psyn_shell-t> set rc_output_threshold_pct_fall 55
```

SEE ALSO

rc_input_threshold_pct_rise(3)
rc_input_threshold_pct_fall(3)
rc_output_threshold_pct_fall(3)
rc_output_threshold_pct_rise(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_rise(3)
lib_thresholds_per_lib(3)

rc_slew_upper_threshold_pct_fall

Specifies the threshold voltage that defines the startpoint of the falling slew calculation.

TYPE

float

DEFAULT

80.000000

GROUP

timing_variables

DESCRIPTION

Specifies the threshold voltage that defines the startpoint of the falling slew calculation. The value is a percent of the voltage source. Allowed values are 0.0 - 100.0 inclusive; the default is 80.0.

This variable is one of 8 variables, listed in Table 1, that affect delay and transition time computations for detailed RC networks. These variables interpret the cell delays and transition times from the Synopsys library. The values specified by these variables are overridden by trippoint values specified in the library so normally it should not be necessary to set the variables. The values specified are only applied to libraries that do not contain trippoint specifications.

Table 1

Variable Name	**Default**
rc_slew_lower_threshold_pct_rise	**20.0**
rc_slew_lower_threshold_pct_fall	**20.0**
rc_slew_upper_threshold_pct_rise	**80.0**
rc_slew_upper_threshold_pct_fall	**80.0**
rc_input_threshold_pct_rise	**50.0**
rc_input_threshold_pct_fall	**50.0**
rc_output_threshold_pct_rise	**50.0**
rc_output_threshold_pct_fall	**50.0**

The default values specify that a cell delay is defined from 50% of the voltage value for the input transition to 50% of the voltage value for the output transition. The default values also specify that a transition time, or slew, is defined from 20% to 80% of the voltage.

To determine the current value of this variable, type **printvar rc_slew_upper_threshold_pct_fall**.

For a list of all timing variables and their current values, type **print_variable_group timing**.

EXAMPLES

The following example specifies that cell delays from the Synopsys library are computed from 50% of the input transition to 55% of the output transition. In addition, the example specifies that transition times in the Synopsys library represent the delay from 10% to 90% of the voltage source.

psyn_shell-t> **set rc_slew_lower_threshold_pct_fall 10** psyn_shell-t> **set rc_slew_lower_threshold_pct_rise 10** psyn_shell-t> **set rc_slew_upper_threshold_pct_fall 90** psyn_shell-t> **set rc_slew_upper_threshold_pct_rise 90** psyn_shell-t> **set rc_input_threshold_pct_rise 50** psyn_shell-t> **set rc_input_threshold_pct_fall 50** psyn_shell-t> **set rc_output_threshold_pct_rise 55** psyn_shell-t> **set rc_output_threshold_pct_fall 55**

SEE ALSO

rc_input_threshold_pct_rise(3)
rc_input_threshold_pct_fall(3)
rc_output_threshold_pct_fall(3)
rc_output_threshold_pct_rise(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_rise(3)
lib_thresholds_per_lib(3)

rc_slew_upper_threshold_pct_rise

Specifies the threshold voltage that defines the endpoint of the rising slew calculation.

TYPE

float

DEFAULT

80.000000

GROUP

timing_variables

DESCRIPTION

Specifies the threshold voltage that defines the endpoint of the rising slew calculation. The value is a percent of the voltage source. Allowed values are 0.0 - 100.0 inclusive; the default is 80.0.

This variable is one of 8 variables, listed in Table 1, that affect delay and transition time computations for detailed RC networks. These variables interpret the cell delays and transition times from the Synopsys library. The values specified by these variables are overridden by trippoint values specified in the library so normally it should not be necessary to set the variables. The values specified are only applied to libraries that do not contain trippoint specifications.

Table 1

Variable Name	**Default**
rc_slew_lower_threshold_pct_rise	**20.0**
rc_slew_lower_threshold_pct_fall	**20.0**
rc_slew_upper_threshold_pct_rise	**80.0**
rc_slew_upper_threshold_pct_fall	**80.0**
rc_input_threshold_pct_rise	**50.0**
rc_input_threshold_pct_fall	**50.0**
rc_output_threshold_pct_rise	**50.0**
rc_output_threshold_pct_fall	**50.0**

The default values specify that a cell delay is defined from 50% of the voltage value for the input transition to 50% of the voltage value for the output transition. The default values also specify that a transition time, or slew, is defined from 20% to 80% of the voltage.

To determine the current value of this variable, type **printvar rc_slew_upper_threshold_pct_rise**.

For a list of all timing variables and their current values, type **print_variable_group timing**.

EXAMPLES

The following example specifies that cell delays from the Synopsys library are computed from 50% of the input transition to 55% of the output transition. In addition, the example specifies that transition times in the Synopsys library represent the delay from 10% to 90% of the voltage source.

```
psyn_shell-t> set rc_slew_lower_threshold_pct_fall 10 psyn_shell-t> set
rc_slew_lower_threshold_pct_rise 10 psyn_shell-t> set
rc_slew_upper_threshold_pct_fall 90 psyn_shell-t> set
rc_slew_upper_threshold_pct_rise 90 psyn_shell-t> set rc_input_threshold_pct_rise 50
psyn_shell-t> set rc_input_threshold_pct_fall 50 psyn_shell-t> set
rc_output_threshold_pct_rise 55 psyn_shell-t> set rc_output_threshold_pct_fall 55
```

SEE ALSO

rc_input_threshold_pct_rise(3)
rc_input_threshold_pct_fall(3)
rc_output_threshold_pct_fall(3)
rc_output_threshold_pct_rise(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_rise(3)
lib_thresholds_per_lib(3)

read_db_lib_warnings

Indicates that warnings are to be printed while a technology .db library is being read in with the **read** command. When false (the default), no warnings are given.

TYPE

Boolean

DEFAULT

false

GROUP

io_variables

DESCRIPTION

Indicates that warnings are to be printed while a technology .db library is being read in with the **read** command. When false (the default), no warnings are given.

To determine the current value of this variable, type **printvar read_db_lib_warnings**. For a list of all **io** variables and their current values, type **print_variable_group io**.

SEE ALSO

read (2); **io_variables** (3).

read_only_attributes

Contains informational attributes, which the user cannot set.

DESCRIPTION

Contains informational attributes. A "read-only" attribute cannot be set by the user.

To determine the value of an attribute, use the **get_attribute** command.

For information on all attributes, refer to the **attributes** manual page.

Read-only Attributes

design_type
: Indicates the current state of the design and has the value *fsm* (finite state machine), *pla* (programmable logic array), *equation* (Boolean logic), or *netlist* (gates). This attribute cannot be set by the user.

is_black_box
: *true* if the reference is not yet linked to a design. This attribute cannot be set by the user.

is_combinational
: *true* if all cells of a design and all designs in its hierarchy are combinational. A cell is combinational if it is non-sequential or non-tristate and all of its outputs compute a combinational logic function. The **report_lib** command will report such a cell as not a black-box. This attribute cannot be set by the user.

is_dw_subblock
: *true* if the object (a cell, a reference, or a design) is a DW subblock that was automatically elaborated. This attribute cannot be set by the user. **Note**: DW subblocks that are manually elaborated will not have this attribute.

is_hierarchical
: *true* if the design contains leaf cells or other levels of hierarchy. This attribute is read-only and cannot be set by the user.

is_mapped
: *true* if all the non-hierarchical cells of a design are mapped to cells in a technology library. This attribute cannot be set by the user.

is_sequential
: *true* if any cells of a design or designs in its hierarchy are sequential. A cell is sequential if it is not combinational. This attribute cannot be set by the user.

is_test_circuitry
: Set by **insert_dft** on the scan cells and nets added to a design during the addition of test circuitry. This attribute cannot be set by the user.

is_synlib_module
: *true* if the object (a cell, a reference, or a design) refers to an unmapped module reference or if the object is (or refers to) a design that was automatically elaborated from a synlib module or a synlib operator. This attribute cannot be set by the user.
Note: synlib modules that are manually elaborated will not have this attribute.

is_synlib_operator
: *true* if the object (a cell or a reference) is a synthetic library operator reference. This attribute cannot be set by the user.

is_unmapped
: *true* if any of the cells are not linked to a design or mapped to a technology library. This attribute cannot be set by the user.

pin_direction
: Direction of a pin. Value can be *in*, *out*, *inout*, or *unknown*. This attribute cannot be set by the user.

port_direction
: Direction of a port. Value can be *in*, *out*, *inout*, or *unknown*. This attribute cannot be set by the user.

ref_name
: The reference name of a cell. This attribute cannot be set by the user.

SEE ALSO

get_attribute(2)
attributes(3)

read_translate_msff

Indicates (when *true*, the default) that master-slave flip-flops (specified with the clocked_on_also syntax) are to be automatically translated to master-slave latches. When *false*, both master and slave remain flip-flops.

TYPE

Boolean

DEFAULT

true

GROUP

io_variables

DESCRIPTION

Indicates (when *true*, the default) that master-slave flip-flops (specified with the clocked_on_also syntax) are to be automatically translated to master-slave latches. When *false*, both master and slave remain flip-flops.

This variable is used during execution of the **read** command, while a technology .db library is being read in by dc_shell and during execution of the **read_lib** command while a technology library is being read in by Library Compiler. The technology .db library is affected only if the program reports that the .db library is being updated and asks you to save the results. Library Compiler always follows this variable during processing.

To determine the current value of this variable, type **printvar read_translate_msff**. For a list of all **io** variables and their current values, type **print_variable_group io**.

SEE ALSO

read_lib(2)
io_variables(3)

reference_attributes

Contains attributes that can be placed on a reference.

DESCRIPTION

Contains attributes that can be placed on a reference.

Several commands exist that can be used to set attributes; however, most attributes can be set by using the **set_attribute** command. If the attribute definition specifies a **set** command, use it to set the attribute; otherwise, use **set_attribute**. If an attribute is read-only, the you cannot set it.

To determine the value of an attribute, use the **get_attribute** command. To remove attributes, use the **remove_attribute** command.

For a more detailed explanation of an attribute, see the manual page of the appropriate **set** command. For information on all attributes, refer to the **attributes** manual page.

Reference Attributes

dont_touch
: Specifies that designs linked to a reference with this attribute are excluded from optimization. Valid values are true (the default) or false. Designs linked to a reference by using the **dont_touch** attribute set to true are not modified or replaced during compile. Set this by using the **set_dont_touch** attribute.

is_black_box
: This is set to true if the reference is not yet linked to a design. This attribute is read-only and you cannot set it.

is_combinational
: This is set to true if all the cells of the referenced design are combinational. A cell is combinational if it is nonsequential or non-tristate and all of its outputs compute a combinational logic function. The **report_lib** command reports such a cell as not a black-box. This attribute is read-only and you cannot set it.

is_dw_subblock
: This is set to true if the object (a cell, a reference, or a design) is a DesignWare subblock that was automatically elaborated. This attribute is read-only and you cannot set it.
Note: DesignWare subblocks that are manually elaborated do not have this attribute.

is_hierarchical
: This is set to true if the design contains leaf cells or other levels of hierarchy. This attribute is read-only and you cannot set it.

is_mapped
: This is set to true if the reference is linked to a design, and all the non-

hierarchical cells of the referenced design are mapped to cells in a technology library. This attribute is read-only and you cannot set it.

is_sequential

This is set to true if all the cells of the referenced design are sequential. A cell is sequential if it is not combinational (if any of its outputs depend on previous inputs). This attribute is read-only and you cannot set it.

is_synlib_module

This is set to true if the object (a cell, a reference, or a design) refers to an unmapped module reference, or the object is (or refers to) a design that was automatically elaborated from a synlib module or a synlib operator. This attribute is read-only and you cannot set it.
Note: synlib modules that are manually elaborated do not have this attribute.

is_synlib_operator

This is set to true if the object (a cell or a reference) is a synthetic library operator reference. This attribute is read-only and you cannot set it.

is_unmapped

This is set to true if any of the non-hierarchical cells of the referenced design are not mapped to cells in a technology library, or the reference is not yet linked to a design. This attribute is read-only and you cannot set it.

scan

When *true*, specifies that cells of the referenced design are always replaced by equivalent scan cells. When *true*, specifies that cells of the referenced design are always replaced by equivalent scan cells during insert_dft. When false, cells are not replaced. Set by using the **set_scan_replacement**.

scan_chain

Includes the specified cells of the referenced design in the scan-chain whose index is the value of this attribute.

ungroup

Specifies that all designs linked to a reference with this attribute are ungrouped (levels of hierarchy represented by these design cells are removed) during compile. Set by using the **set_ungroup** command.

SEE ALSO

get_attribute(2)
insert_dft(2)
remove_attribute(2)
set_attribute(2)
set_scan_replacement(2)
attributes(3)

register_duplicate

Controls whether **compile** should invoke **Register duplication** or not.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

The **register_duplicate** variable (when set to true), duplicates the high fanout registers to reduce the number of fanouts for each register.

Fanouts for registers that exceed a certain limit can cause an adverse affect on the circuit performance. Currently, in *Design Compiler*, the fanout is controlled only by buffer insertion. Some FPGA architectures can lack buffers in the routing architecture that can cause the fixed fanout limit to exceed, which can cause an error in the back-end tool. To address this problem, the **register_duplicate** variable provides a method to duplicate registers in an optimized way.

EXAMPLES

```
prompt> set_register_max_fanout 25 top
prompt> register_duplicate = true
prompt>set_attribute xfpga_virtex2-5 -type float default_fanout_load 100
Compile will indicate register duplication status as :-


   ELAPSED             WORST NEG   TOTAL NEG  DESIGN

    TIME      AREA      SLACK       SLACK      RULE COST         ENDPOINT

  --------- --------- --------- --------- --------- -------------------------

Information: Duplicating register r1_reg with fanout load of 50.00 (REGDUP-3)



  Optimization Complete
```

SEE ALSO

`compile(2)`

reoptimize_design_changed_list_file_name

Creates a file in which to store the list of cells that changed and cells and nets that were added during post-layout or in-place optimization.

TYPE

string

DEFAULT

""

GROUP

compile_variables
links_to_layout_variables

DESCRIPTION

Creates a file in which to store the list of cells that changed and cells and nets that were added during post-layout or in-place optimization.

When the **reoptimize_design_changed_list_file_name** variable is set to a particular file name, the **reoptimize_design -in_place** and **reoptimize_design -post_layout_opto** commands output to the file a list of design modifications and additions. The commands use the file to store the list of those cells that changed and those cells and nets that were added. By default, this variable is set to an empty string, and no such file is created during post-layout or in-place optimization.

For backwards compatibility, the **compile -in_place** command also sends the list of changes to this file.

If the specified file already exists, the new changes and additions are appended to the existing file.

During in-place optimization, cells are not normally added. However, if the **compile_ok_to_buffer_during_inplace_opt** flag is asserted, in-place optimization might create new cells. If it does, those cells are included in the specified change-list file. New nets are also included in the file.

During post-layout optimization, net changes are not tracked. Consequently, a cell is considered changed if its library cell has changed. This is different from in-place optimization, where a cell is considered changed if any of its connected nets are changed.

To determine the current value of this variable, use **printvar reoptimize_design_changed_list_file_name**. For a list of all **compile** variables and their current values, use the **print_variable_group compile** command. For a list of all **links_to_layout** variables and their current values, use the **print_variable_group links_to_layout** command.

SEE ALSO

compile (2), **reoptimize_design** (2); **compile_ok_to_buffer_during_inplace_opt** (3), **compile_variables** (3), **links_to_layout_variables** (3).

report_default_significant_digits

Sets the default number of significant digits for many reports.

TYPE

integer

DEFAULT

-1

GROUP

none

DESCRIPTION

The **report_default_significant_digits** variable sets the default number of significant digits for many reports. Allowed values are 0-13; the default is -1. A default of -1 indicates that command specific default precision value will be used for reporting. Some report commands (for example, **report_timing**, **report_cell**) have a **-significant_digits** option, which overrides the value of this variable.

Not all reports respond to this variable. Check the man pages for individual reports to determine whether they support this feature.

To determine the current value of this variable, type **printvar report_default_significant_digits** or **printvar report_default_significant_digits**.

Once set, the value of the variable will be used by the subsequent reporting commands, if command specific **-significant_digits** option is not used. The value of the variable can be reset. To do so, set the value of the variable to -1. Doing so, command specific default will be used as precision for reporting if **-significant_digits** on the command is not used.

For example, command specific default precision of **report_cell** is 6. If **report_default_significant_digits** is set to 5 and **report_cell** issued thereafter, a precision of 5 will be used during **report_cell** report. If **report_cell -significant_digits 3** is issued after the above variable setting, a precision of 3 will be followed for **report_cell** report. To view the **report_cell** report with the command default precision of 6, reset the value of the variable **report_default_significant_digits**, by setting it to -1.

If an invalid value is set(not in the range 0-13 and not -1), an error will be issued and the previous valid value will be restored. Example below illustrates the usage of the variable.

```
prompt> list report_default_significant_digits
report_default_significant_digits = -1
prompt> report_default_significant_digits = 4
```

```
4
prompt> list report_default_significant_digits
report_default_significant_digits = 4
prompt> report_default_significant_digits = -44
Error: can’t set report_default_significant_digits: must be in range 0 to 13
        Use error_info for more info. (CMD-013)
4
prompt> list report_default_significant_digits
report_default_significant_digits = 4
```

SEE ALSO

```
report_timing(2)
report_clock_gating_check(2)
report_path_budget(2)
report_constraint(2)
write_sdf(2)
report_cell(2)
report_net(2)
report_qor(2)
```

rom_auto_inferring

Inferring ROM from RTL description.

SYNTAX

Boolean **rom_auto_inferring** *true/false*

TYPE

Boolean

DEFAULT

true

GROUP

DESCRIPTION

When this variable is *true*, the tool will try to inferring ROM from RTL description.

SEE ALSO

route_guide_attributes

Contains attributes related to route guide.

DESCRIPTION

Contains attributes related to route guide.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified object. Specified with **list_attribute -class route_guide -application**, the definition of attributes can be listed.

Route Guide Attributes

affects

    Specifies the affects of a route guide, which is route.
    The data type of **affects** is string.
    This attribute is read-only.

area

    Specifies area of a route guide.
    The data type of **area** is float.
    This attribute is read-only.

bbox

    Specifies the bounding-box of a route guide. The **bbox** is represented by a **rectangle**.
    The format of a *rectangle* specification is {{llx lly} {urx ury}}, which specifies the lower-left and upper-right corners of the rectangle.
    The data type of **bbox** is string.
    This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

cell_id

    Specifies Milkyway design ID in which a route guide object is located.
    The data type of **cell_id** is integer.
    This attribute is read-only.

horizontal_track_utilization

    Specifies the horizontal track utilization for the route guide.
    The data type of **horizontal_track_utilization** is integer.
    This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

layer

    Specifies layer name on which a route guide is.
    The data type of **layer** is string.
    This attribute is read-only.

layer_number

    Specifies layer number on which a route guide is.
    The data type of **layer_number** is integer.

This attribute is read-only.

name
: Specifies name of a route guide object.
The data type of **name** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

no_preroute_layers
: Specifies the layers that cannot contain preroutes.
The data type of **no_preroute_layers** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

no_signal_layers
: Specifies the layers that cannot contain signals.
The data type of **no_signal_layers** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

object_class
: Specifies object class name of a route guide, which is **route_guide**.
The data type of **object_class** is string.
This attribute is read-only.

object_id
: Specifies object ID in Milkyway design file.
The data type of **object_id** is integer.
This attribute is read-only.

object_type
: Specifies geometry type of a route guide, which can be RECTANGLE or POLYGON.
The data type of **object_type** is string.
This attribute is read-only.

points
: Specifies point list of a route guide’s boundary.
The data type of **points** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

preferred_direction_only_layers
: Specifies the layers that cannot make nonPreferredDirection wires.
The data type of **preferred_direction_only_layers** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

repair_as_single_sbox
: Specifies whether this route guide should be repaired as a single sbox when there is a difficult violation on a prerouted wire or inside a large macro.
The data type of **repair_as_single_sbox** is boolean.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

switch_preferred_direction

Specifies whether to switch the preferred direction for the route guide.
The data type of **switch_preferred_direction** is boolean.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

type

Specifies the affects of a route guide, which is route.
The data type of **type** is string.
This attribute is read-only.

vertical_track_utilization

Specifies the vertical track utilization for the route guide.
The data type of **vertical_track_utilization** is integer.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

zero_min_spacing

Specifies whether zero minimum spacing is allowed for the route guide.
The data type of **zero_min_spacing** is boolean.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

SEE ALSO

get_attribute(2),
list_attribute(2),
report_attribute(2),
set_attribute(2).

rp_shift_column_for_fixed_cells

Determines whether the tool allows the horizontal shifting of relative placement columns in the presence of single tap cells.

TYPE

Boolean

DEFAULT

false

GROUP

physopt

DESCRIPTION

The value you specify for the **rp_shift_column_for_fixed_cells** variable determines whether the tool performs horizontal relative placement column shifting for single tap cells. The default value of this variable is **false**. The default behavior is to horizontal shift the relative placement column only when a tap cell array is present, not for a single tap cell.

To see the current value of this variable, type **printvar rp_shift_column_for_fixed_cells**.

rtl_load_resistance_factor

Specifies a factor to be used by the **set_rtl_load** command to calculate resistance values from capacitance values for RTL loads.

TYPE

float

DEFAULT

0.0

GROUP

compile_variables

DESCRIPTION

Specifies a factor to be used by the **set_rtl_load** command to calculate resistance values from capacitance values for RTL loads.

You do not need to specify resistance values directly to **set_rtl_load**. Instead, you can cause the resistance value to be calculated as a constant factor times capacitance by setting the **rtl_load_resistance_factor** variable to the constant factor required. Then, you execute **set_rtl_load** with only the **-capacitance** option, and the command calculates the resistance from the specified capacitance using the constant factor.

For example, if you set the **rtl_load_resistance** variable to 0.5, specify the rtl-load capacitance of a pin as 4, and do not specify the rtl-load resistance, **set_rtl_load** calculates the rtl-load resistance to be 2. The factor is applied to each annotated pin of a net individually, not to the net's capacitance as a whole. Note that the default library units are used throughout.

To determine the current value of this variable, use **printvar rtl_load_resistance_factor**.

SEE ALSO

remove_rtl_load(2)
set_rtl_load(2)

schematic_variables

Variables that affect **create_schematic**.

SYNTAX

string **bus_dimension_separator_style** =   "]["
string **bus_naming_style** =   "%s[%d]"
string **bus_range_separator_style** =   ":"
string **default_schematic_options** = "-size infinite"
Boolean **duplicate_ports** = "false"
Boolean **gen_bussing_exact_implicit** = "false"
string **gen_cell_pin_name_separator** = "/"
Boolean **gen_create_netlist_busses** =   "false"
Boolean **gen_dont_show_single_bit_busses** = "false"
Boolean **gen_match_ripper_wire_widths** = "false"
integer **gen_max_compound_name_length**   256
integer **gen_max_ports_on_symbol_side** = 0
string **gen_open_name_prefix** = "Open" ;
string **gen_open_name_postfix** = "" ;
Boolean **gen_show_created_busses** = "false"
Boolean **gen_show_created_symbols** = "false"
Boolean **gen_single_osc_per_name** = "false"
string **generic_symbol_library** = "generic.sdb"
Boolean **single_group_per_sheet** = "false"
string **sort_outputs** =   "false"
string **symbol_library** = {"your_library.sdb"}
Boolean **use_port_name_for_oscs** = "true"

DESCRIPTION

These variables affect **create_schematic**. Defaults are specified above, under Syntax.

For a list of **schematic** variables and their current values, type **print_variable_group schematic**. To view this manual page online, type **help schematic_variables**. To view an individual variable description, type **help *var***, where *var* is the name of the variable.

bus_dimension_separator_style
: This variable affects the **read** command with the **verilog, vhdl or edif** format options. In conjunction with the **bus_naming_style** variable, **bus_dimension_separator_style** controls naming of individual bit-blasted ports derived from multi-dimensional arrays. This variable also affects the naming of bit-blasted multi-dimensional instance arrays and bit-blasted multi-dimensional net arrays in exactly the same way as for ports.

bus_naming_style
: This variable affects the **read**, **write**, and **create_schematic** commands. For the **read** command with the **vhdl** or **verilog** format options, this variable controls naming of bit-blasted ports in **dc_shell** when they are created from multi-dimensional busses in the original source.
For the **write** command with the **edif** format option, this variable controls naming of individual members of each bus in the edif output. Also, together

with the **bus_range_separator_style** variable, this variable controls how ranged names of busses are written to the edif output.
For the **create_schematic** command, this variable controls naming of bus rippers that rip off bits from bussed nets. Together with the **bus_range_separator_style** variable, it also controls naming of bussed nets, bussed ports, and bussed rippers in the schematic. Works the same as for **write -f edif**.

bus_range_separator_style

This variable affects the **write** and **create_schematic** commands. For the **write** command with the **edif** format option, this variable is used together with the **bus_naming_style** variable to generate the style for naming bus ranges.
For the **create_schematic** command, this variable, together with the **bus_range_separator_style** variable, also controls naming of bussed nets, bussed ports and bussed rippers in the schematic. Works the same as for **write -f edif**.

default_schematic_options

Specifies options to use when schematics are generated. When set to *-size infinite* (the default value), the schematic for a design is displayed on a single page. Used by the Design Analyzer.

duplicate_ports

A partitioning option that determines whether ports are to be drawn on every sheet for which an input or output signal appears. When *true*, no off-sheet connectors are used for input and output signals, and signal ports are duplicated where indicated on each sheet. When *false* (the default value), off-sheet connectors are used and signal ports are not duplicated.

gen_bussing_exact_implicit

When *true*, specifies that schematics generated with the **-implicit** option should contain no bus rippers. All bussed connections should be shown with implicit bus names. The default is *false*.

gen_cell_pin_name_separator

A value that is to be used to separate the cell and pin names in the bus names generated by **create_schematic**. By default, "/" is used to separate the cell and pin names.

gen_create_netlist_busses

When *true*, **create_schematic** will create netlist busses whenever it creates busses on the schematic. Usually this happens when **create_schematic** creates busses to connect to bussed pins on the schematic, but it may also happen when there are bussed ports on the schematic. The default is *false*.

gen_dont_show_single_bit_busses

When this variable is *true* and the **gen_show_created_busses** variable is also *true*, the names of single bit busses are not printed. Only schematic busses that have more than one bit are printed. The default is *false.*

gen_match_ripper_wire_widths

When *true*, specifies that **create_schematic** generates rippers such that the width of the ripper always equals the width of the ripped net. Any rippers whose wire ends are connected to scalar nets will be of unit width. The default is *false*.

gen_max_compound_name_length
: Controls the maximum length for compound names of bus bundles for **create_schematic -sge**. Any busses with names longer than the maximum length are decomposed into their individual members in the schematic.

gen_max_ports_on_symbol_side
: Specifies the maximum allowed size of a symbol created by **create_schematic**.

gen_open_name_prefix
: Specifies the prefix to be used by **create_schematic -sge** when creating placeholder net names for unconnected pins. The default is "Open".
The format of the net names is "%s%d%s", where the first "%s" is replaced by the value of **gen_open_name_prefix**, the second "%s" is replaced by the value of **gen_open_name_postfix**, and the "%d" is replaced by an integer whose value is generated automatically by **create_schematic -sge**.

gen_open_name_postfix
: Specifies the postfix to be used by **create_schematic -sge** when creating placeholder net names for unconnected pins. The default is "".
The format of the net names is "%s%d%s", where the first "%s" is replaced by the value of **gen_open_name_prefix**, the second "%s" is replaced by the value of **gen_open_name_postfix**, and the "%d" is replaced by an integer whose value is generated automatically by **create_schematic -sge**.

gen_show_created_busses
: When *true*, a message is printed out every time a schematic bus is created from cell pins for which no equivalent net bus exists in the netlist. The default is *false*.

gen_show_created_symbols
: When *true*, **create_schematic** prints a warning message every time it generates a new symbol for a cell because an appropriate symbol could not be found in the symbol libraries. The default is *false*.

gen_single_osc_per_name
: When *true*, in cases where there could potentially be more than one off sheet connector with the same name on any schematic page, only one connector is drawn and the others just have their net segments appear as unconnected stubs. The default is *false*.

generic_symbol_library
: The name of the **db** file that contains generic symbols, templates, and layers used for schematics. The default is *generic.sdb*.

single_group_per_sheet
: A partitioning option that, when *true*, specifies that only one logic group is put on a sheet. This eliminates the possibility of more than one off-sheet connector with the same name on a single sheet. The default is *false*.

sort_outputs
: When *true*, sorts output ports on the schematic by port name. The default is *false*.

symbol_library
: A list of symbol library names to use during schematic generation.

use_port_name_for_oscs

    A partitioning option that, when *true* (the default value), specifies that off-sheet connectors for nets that also have ports on them are given the name of the port. When *false*, the connectors are given the name of the net.

SEE ALSO

create_schematic (2), **read_lib** (2); **view_variables** (3).

sdc_write_unambiguous_names

Ensures that cell, net, pin, lib_cell, and lib_pin names that are written to the SDC file are not ambiguous.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Ensures that cell, net, pin, lib_cell, and lib_pin names that are written to the SDC file are not ambiguous. The default value is *true*.

When hierarchy has been partially flattened, embedded hierarchy separators can make names ambiguous. It is not clear which hierarchy separator characters are part of the name and which are real separators.

Beginning with SDC Version 1.2, hierarchical names can be made nonambiguous using the **set_hierarchy_separator** SDC command and/or the **-hsc** option on the **get_cells**, **get_lib_cells**, **get_lib_pins**, **get_nets**, and **get_pins** SDC object access commands. By default, PrimeTime and Design Compiler write an SDC file, using these features to create nonambiguous names.

It is wise to write SDC files that contain names that are not ambiguous. However, if you are using a third-party application that does not fully support SDC 1.2 or later versions (that is, it does not support the nonambiguous hierarchical names features of SDC), you can suppress these features by setting the variable **sdc_write_unambiguous_names** to *false*. The **write_sdc** command issues a warning if you have set this variable to *false*.

To determine the current value of this variable, use **printvar sdc_write_unambiguous_names**.

SEE ALSO

printvar(2)
write_sdc(2)

sdfout_allow_non_positive_constraints

Writes out PATHCONSTRAINT constructs with nonpositive (<= 0) constraint values. This variable will be obsolete in the next release. Please adjust your scripts accordingly.

TYPE

Boolean

DEFAULT

false

GROUP

links_to_layout_variables

DESCRIPTION

Writes out PATHCONSTRAINT constructs with nonpositive (<= 0) constraint values. When *true*, **write_constraints -format sdf** writes out PATHCONSTRAINT constructs with nonpositive (<= 0) constraint values. When *false* (the default), paths with nonpositive constraints are written with a constraint value of 0.01.

Nonpositive constraints can occur when the arrival time at a path startpoint is larger than the required arrival time at the path endpoint. This typically indicates an error, but is sometimes valid when generating constraints for a subdesign.

To determine the current value of this variable, type **printvar sdfout_allow_non_positive_constraints**. For a list of all **links_to_layout** variables and their current values, type **print_variable_group links_to_layout**.

SEE ALSO

write_constraints(2)
links_to_layout_variables(3)

sdfout_min_fall_cell_delay

Specifies the minimum non-back-annotated fall cell delay that the **write_timing** command writes to a timing file in SDF format. This variable will be obsolete in the next release. Please adjust your scripts accordingly.

TYPE

float

DEFAULT

0.000000

GROUP

io_variables

DESCRIPTION

Specifies the minimum non-back-annotated fall cell delay that the **write_timing** command writes to a timing file in SDF format. The value of this variable can be *positive*, *negative*, or *zero* (the default); the unit must be the same as the timing unit in the technology library.

By default, if this variable is not set, **write_timing** writes to the SDF file all fall cell delay values greater than or equal to zero. You can override this default behavior for non-back-annotated delays by setting **sdfout_min_fall_cell_delay** to a minimum value; **write_timing** does not write values that are less than this minimum. However, you cannot override the default behavior for back-annotated delays; **write_timing** always writes values of delays that have been back-annotated, regardless of the value of this variable.

Use this variable to filter non-back-annotated delays so that **write_timing** writes only values that are significant (greater than the specified minimum value). Also, if you do not want non-annotated fall cell delays to be written to the SDF file, set this variable to a value higher than any of the non-annotated cell delays in the design.

To determine the current value of this variable, use **printvar sdfout_min_fall_cell_delay**. For a list of all **io_variables** and their current values, use the **print_variable_group io** command.

SEE ALSO

write_timing(2)
sdfout_min_rise_cell_delay(3)

sdfout_min_fall_net_delay

Specifies the minimum non-back-annotated fall net delay that **write_timing** can write to a timing file in SDF format. This variable will be obsolete in the next release. Please adjust your scripts accordingly.

TYPE

float

DEFAULT

0.000000

GROUP

io_variables

DESCRIPTION

Specifies the minimum non-back-annotated fall net delay that **write_timing** can write to a timing file in SDF format. The value of this variable can be *positive*, *negative*, or *zero* (the default); the unit must be the same as the timing unit in the technology library.

By default, if this variable is not set, **write_timing** writes to the SDF file all fall net delay values greater than or equal to zero. You can override this default behavior for non-back-annotated delays by setting **sdfout_min_fall_net_delay** to a minimum value; **write_timing** does not write values that are less than this minimum. However, you cannot override the default behavior for back-annotated delays; **write_timing** always writes values of delays that have been back-annotated, regardless of the value of this variable.

Use this variable to filter non-back-annotated delays so that **write_timing** writes only values that are significant (greater than the specified minimum value). Also, if you do not want any non-annotated fall net delays to be written to the SDF file, set this variable to a value higher than any of the non-annotated net delays in the design.

To determine the current value of this variable, use **printvar sdfout_min_fall_net_delay**. For a list of all **io_variables** and their current values, use the **print_variable_group io** command.

SEE ALSO

write_timing(2)
sdfout_min_rise_net_delay(3)

sdfout_min_rise_cell_delay

Specifies the minimum non-back-annotated rise cell delay that **write_timing** can write to a timing file in SDF format. This variable will be obsolete in the next release. Please adjust your scripts accordingly.

TYPE

float

DEFAULT

0.000000

GROUP

io_variables

DESCRIPTION

Specifies the minimum non-back-annotated rise cell delay that **write_timing** can write to a timing file in SDF format. The value of this variable can be *positive*, *negative*, or *zero* (the default); the unit must be the same as the timing unit in the technology library.

By default, if this variable is not set, **write_timing** writes to the SDF file all rise cell delay values greater than or equal to zero. You can override this default behavior for non-back-annotated delays by setting **sdfout_min_rise_cell_delay** to a minimum value; **write_timing** will not write values less than this minimum. However, you cannot override the default behavior for back-annotated delays; **write_timing** always writes values of delays that have been back-annotated, regardless of the value of this variable.

Use this variable to filter non-back-annotated delays so that **write_timing** writes only values that are significant (greater than the specified minimum value). Also, if you do not want any non-annotated rise cell delays to be written to the SDF file, set this variable to a value higher than any of the non-annotated cell delays in the design.

To determine the current value of this variable, use **printvar sdfout_min_rise_cell_delay**. For a list of all **io_variables** and their current values, use the **print_variable_group io** command.

SEE ALSO

write_timing(2)
sdfout_min_fall_cell_delay(3)

sdfout_min_rise_net_delay

Specifies the minimum non-back-annotated rise net delay that the **write_timing** command can write to a timing file in SDF format. This variable will be obsolete in the next release. Please adjust your scripts accordingly.

TYPE

float

DEFAULT

0.000000

GROUP

io_variables

DESCRIPTION

Specifies the minimum non-back-annotated rise net delay that the **write_timing** command can write to a timing file in SDF format. The value of this variable can be *positive*, *negative*, or *zero* (the default); the unit must be the same as the timing unit in the technology library.

By default, if this variable is not set, **write_timing** writes to the SDF file all rise net delay values greater than or equal to zero. You can override this default behavior for non-back-annotated delays by setting **sdfout_min_rise_net_delay** to a minimum value; **write_timing** does not write values less than this minimum. However, you cannot override the default behavior for back-annotated delays; **write_timing** always writes values of delays that have been back-annotated, regardless of the value of this variable.

Use this variable to filter non-back-annotated delays so that **write_timing** writes only values that are significant (greater than the specified minimum value). Also, if you do not want any non-annotated rise net delays to be written to the SDF file, set this variable to a value higher than any of the non-annotated net delays in the design.

To determine the current value of this variable, use **printvar sdfout_min_rise_net_delay**. For a list of all **io_variables** and their current values, use the **print_variable_group io** command.

SEE ALSO

write_timing(2)
sdfout_min_fall_net_delay(3)

sdfout_time_scale

Specifies the time scale of the delays written to timing files in SDF format. This variable will be obsolete in the next release. Please adjust your scripts accordingly.

TYPE

float

DEFAULT

1.000000

GROUP

io_variables

DESCRIPTION

Specifies the time scale of the delays written to timing files in SDF format. Delays from Design Compiler are written to timing files with **write_timing**. The **sdfout_time_scale** variable must be set if the library has no time unit specified and if the time unit of the delays in the library is different than 1 nanosecond. By default, the time unit is nanosecond and the time scale is 1. The only valid values for the SDF format are 0.001, 0.01, 0.1, 1, 10 and 100. The time unit is specified in the library with the attributes **time_scale** and **time_unit_name**.

For example, a library with timing values in 10 picoseconds is specified with the attributes:

```
time_scale = 10 and time_unit_name = ps
```

When the attribute **time_scale** is missing in the library, use the **sdfout_time_scale** variable to specify the scale of the timing unit. For example, if the library has no time unit specified but is in 10 ns, set **sdfout_time_scale** to 10 in order to specify the time unit as 10 ns.

To determine the current value of this variable use **printvar sdfout_time_scale**. For a list of all **io_variables** and their current values, use the **print_variable_group io** command.

SEE ALSO

write_timing(2)

sdfout_top_instance_name

Specifies the name prepended to all instance names when writing timing files in SDF format. This variable will be obsolete in the next release. Please adjust your scripts accordingly.

TYPE

string

DEFAULT

""

GROUP

io_variables

DESCRIPTION

Specifies the name prepended to all instance names when writing timing files in SDF format. Timing files are written with the **write_timing** command. By default **write_timing** prepends no name to all cell instance names. Set this variable when you want the cell instance names to contain a prepended name.

For example set **sdfout_top_instance_name = "stim.cell1"** if the timing file should contain:

```
(DESIGN "fifo")
(DIVIDER .)
(CELL
   (CELLTYPE "fifo")
   (INSTANCE stim.cell1)
)
(CELL
   (CELLTYPE "AND2")
   (INSTANCE stim.cell1.U1)
```

With the previous example, if **sdfout_top_instance_name = ""** timing file will contain:

```
(DESIGN "fifo")
(DIVIDER .)
(CELL
   (CELLTYPE "fifo")
   (INSTANCE )
)
(CELL
   (CELLTYPE "AND2")
   (INSTANCE U1)
```

To determine the current value of this variable, use **printvar sdfout_top_instance_name**. For a list of all **io_variables** variables and their current values, use the **print_variable_group io** command.

SEE ALSO

read_timing(2)
sdfin_top_instance_name(3)

sdfout_write_to_output

Specifies whether the **write_timing -f sdf** command writes interconnect delays between cells and top-level output ports. This variable will be obsolete in the next release. Please adjust your scripts accordingly.

TYPE

Boolean

DEFAULT

false

GROUP

io_variables

DESCRIPTION

Specifies whether the **write_timing -f sdf** command writes interconnect delays between cells and top-level output ports. The **sdfout_write_to_output** variable also determines whether output to output pin IOPATH statements are written for cells that contain output-to-output timing arcs. v1.0 SDF does not support output-to-output timing for either IOPATH or INTERCONNECT statements. However, the Synopsys Simulator does support output-to-output timing for these statements.

Set this variable to *true*, if the targeted SDF reader is the Synopsys Simulator.

Leave this variable set to *false*, the default, to ensure that generated SDF files comply with the v1.0 specification.

Set this variable before using **write_timing**. To check the value of this variable, use the command **printvar sdfout_write_to_output**.

SEE ALSO

write_timing(2)

search_path

Specifies directories that the tool searches for files specified without directory names.

TYPE

list

DEFAULT

{*search_path* + .}

GROUP

system_variables

DESCRIPTION

Specifies directories that the tool searches for files specified without directory names. The search includes looking for technology and symbol libraries, design files, and so on. The value of this variable is a list of directory names and is usually set to a central library directory.

To determine the current value of this variable, use **printvar search_path**. For a list of all **system** variables and their current values, use the **print_variable_group system** command.

SEE ALSO

system_variables(3)

sh_command_log_file

Specifies the name of the file to which is written a log of the initial values of variables and executed commands.

TYPE

string

DEFAULT

command.log

DESCRIPTION

Specifies the name of the file to which is written a log of the initial values of variables and executed commands.

The default is command.log. If the value is an empty string, a command log file is not created.

To determine the current value of this variable, use **printvar sh_command_log_file**.

SEE ALSO

printvar(2)
view_command_log_file(3)
view_log_file(3)

sh_enable_line_editing

Enables the command line editing capabilities. This variable is for use in Tcl mode only.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

If set to true it enables advanced UNIX-like shell capabilities. This variable is for use in Tcl mode only.

This variable needs to be set in the .synopsys_dc.setup file to take effect.

Key Bindings

The **sh_list_key_bindings** command displays current key bindings and the edit mode. To change the edit mode, use the **sh_line_editing_mode** variable on the shell.

Command Completion

The editor is able to complete commands, options, variables, and files given a unique abbreviation. You need to type only a part of a word and press the tab key to get the complete command, variable, or file. For command options, type '-' and press the tab key to get the options list.

If no match is found, the terminal bell rings. If the word is already complete a space is added to the end, if it is not already there to speed up typing and provide a visual indicator of successful completion. Completed text pushes the rest of the line to the right. If there are multiple matches then all the matching commands, options, files, or variables are autolisted.

Completion works in following context sensitive way:

The first token of a command line	:	completes commands
Token that begins with "-" after a command	:	completes command arguments
After a ">", "|" or a "sh" command	:	completes filenames
After a set, unset or printvar command	:	completes the variables
After '$' symbol	:	completes the variables
After the help command	:	completes command

```
After the man command                      :  completes commands or variables

Any token which is not the first token
and does not match any of the above rules  :  completes filenames
```

SEE ALSO

`sh_line_editing_mode(3)`

sh_line_editing_mode

Enables vi or emacs editing mode. This variable is for use in Tcl mode only.

TYPE

String

DEFAULT

emacs

DESCRIPTION

This variable can be used to set the command line editor mode to either vi or emacs. Valid values are emacs or vi.

Use **sh_list_key_bindings** command to display the current key bindings and edit mode.

This variable is for use in Tcl mode only.

This variable can be set on the shell or inside .synopsys_dc.setup file to take effect.

SEE ALSO

sh_enable_line_editing(3)

sh_source_uses_search_path

Causes the **search** command to use the **search_path** variable to search for files. This variable is for use in **dc_shell-t** (Tcl mode of dc_shell) only.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Causes the **search** command to use the **search_path** variable to search for files, when **sh_source_uses_search_path** is *true* (the default). When *false*, the **source** command considers this variable's file argument literally. This variable is for use in **dc_shell-t** (Tcl mode of dc_shell) only.

To determine the current value of this variable, use **printvar sh_source_uses_search_path**.

SEE ALSO

printvar(2)
source(2)
search_path(3)

shape_attributes

Contains attributes related to shape.

DESCRIPTION

Contains attributes related to shape.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified object. Specified with **list_attribute -class shape -application**, the definition of attributes can be listed.

Shape Attributes

bbox

Specifies the bounding-box of a shape. The **bbox** is represented by a **rectangle**. The format of a *rectangle* specification is {{llx lly} {urx ury}}, which specifies the lower-left and upper-right corners of the rectangle.
The data type of **bbox** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_ll

Specifies the lower-left corner of the bounding-box of a shape.
The **bbox_ll** is represented by a **point**. The format of a *point* specification is {x y}.
You can get the **attr_name** of a shape, by accessing the first element of its **bbox**.
The data type of **bbox_ll** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_llx

Specifies x coordinate of the lower-left corner of the bounding-box of a shape.
The data type of **bbox_llx** is integer.
This attribute is read-only.

bbox_lly

Specifies y coordinate of the lower-left corner of the bounding-box of a shape.
The data type of **bbox_lly** is integer.
This attribute is read-only.

bbox_ur

Specifies the upper-right corner of the bounding-box of a shape.
The fbbox_ur is represented by a **point**. The format of a *point* specification is {x y}.
You can get the **bbox_ur** of a shape, by accessing the second element of its **bbox**.
The data type of **bbox_ur** is string.
This attribute is writable. You can use **set_attribute** to modify its value on

a specified object.

bbox_urx

Specifies x coordinate of the upper-right corner of the bounding-box of a shape.
The data type of **bbox_urx** is integer.
This attribute is read-only.

bbox_ury

Specifies y coordinate of the upper-right corner of the bounding-box of a shape.
The data type of **bbox_ury** is integer.
This attribute is read-only.

cell_id

Specifies Milkyway design ID in which a shape object is located.
The data type of **cell_id** is integer.
This attribute is read-only.

datatype_number

Specifies GDSII datatype number of a shape object.
The data type of **datatype_number** is integer.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

endcap

Specifies alignment type of a wire or path end.
The data type of **endcap** is string.
Its valid values are:

- **square_ends**
- **round_ends**
- **square_ends_by_half_width**
- **octagon_ends_by_half_width**

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

layer

Specifies layer name of a shape object.
The data type of **layer** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

layer_name

Specifies layer name of a shape object.

The data type of **layer_name** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

layer_number

Specifies layer number of a shape object.
The data type of **layer_number** is integer.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

length

Specifies length of a wire or path object in user units.
The data type of **length** is float.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

name

Specifies name of a shape object.
The data type of **name** is string.
This attribute is read-only.

net_id

Specifies object ID of the net associated with a shape object.
The data type of **net_id** is integer.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

net_type

Specifies type of net associated with a shape object.
The data type of **net_type** is string.
This attribute is read-only.

object_class

Specifies object class name of a shape, which is **shape**.
The data type of **object_class** is string.
This attribute is read-only.

object_id

Specifies object ID in Milkyway design file.
The data type of **object_id** is integer.
This attribute is read-only.

object_type

Specifies object type name, which can be **RECTANGLE**, **POLYGON**, **TRAPEZOID**, **PATH**, **HWIRE**, and **VWIRE**.
The data type of **object_type** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

owner

Specifies Milkyway design file name in which a terminal is located.
The data type of **owner** is string.
This attribute is read-only.

owner_net

Specifies net name which a shape object is connected to.
The data type of **owner_net** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

points

Specifies point list of a shape's boundary.
The data type of **points** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

route_type

Specifies route type of a shape object.
The data type of **route_type** is string.
Its valid values are:

- **User Enter** or **user_enter**
- **Signal Route** or **signal_route**
- **Signal Route (Global)** or **signal_route_global**
- **P/G Ring** or **pg_ring**
- **Clk Ring** or **clk_ring**
- **P/G Strap** or **pg_strap**
- **Clk Strap** or **clk_strap**
- **P/G Macro/IO Pin Conn** or **pg_macro_io_pin_conn**
- **P/G Std. Cell Pin Conn** or **pg_std_cell_pin_conn**
- **Zero-Skew Route** or **clk_zero_skew_route**
- **Bus** or **bus**
- **Shield (fix)** or **shield**

- **Shield (dynamic)** or **shield_dynamic**

- **Fill Track** or **clk_fill_track**

- **Unknown** or **unknown**

It is determined by Tcl variable mw_attr_value_no_space whether **get_attribute** or **report_attribute** returns route_type containing spaces or underscores.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

width

Specifies width of a wire or path object in user units.
The data type of **width** is float.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

SEE ALSO

get_attribute(2),
list_attribute(2),
report_attribute(2),
set_attribute(2).

si_use_partial_grounding_for_min_analysis

Affects the behavior of **report_timing** and **compile** with crosstalk effect is enabled.

TYPE

Boolean

DEFAULT

false

GROUP

si_variables

DESCRIPTION

Affects the behavior, runtime, and CPU usage of **report_timing** and **compile**. When set to *true*, min crosstalk delta delay will be calculated with coupling capacitance partially grounded. When set to *false* (the default value), min crosstalk delta delay will be calculated with coupling capacitance fully grounded. To obtain the more accurate behavior of crosstalk timing analysis, set this variable to *true*.

To determine the current value of this variable, type **printvar fBsi_use_partial_grounding_for_min_analysis**. For a list of all **si** variables and their current values, type **print_variable_group si**.

SEE ALSO

compile(2)
report_timing(2)
si_variables(3)

si_variables

Variables that affect signal integrity analysis.

SYNTAX

Boolean **si_use_partial_grounding_for_min_analysis** = false

DESCRIPTION

These variables directly affect signal integrity analysis in **icc_shell**. Defaults are shown under Syntax.

For a list of si_variables, type **print_variable_group si**. To view an individual variable description, type **help *var***, where *var* is the name of the variable.

si_use_partial_grounding_for_min_analysis

This variable is used to obtain more accurate crosstalk delay delay for min timing analysis. By default, this variable is false. When set to *true*, min crosstalk delta delay will be calculated with coupling capacitance partially grounded.

SEE ALSO

compile (2), **set_si_options** (2), **si_use_partial_grounding_for_min_analysis** (3).

si_xtalk_reselect_delta_and_slack

Reselect nets that satisfy both delta delay and slack reselection criteria.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When true, the intersection of sets of nets reselected by delta delay and slack based criteria is used. For a net to be reselected the following must be true: - The net is reselected by absolute delta delay AND - The net is reselected by relative delta delay AND - The net is reselected by setup OR hold slack

When true, the nets that satisfy only one of the above criteria (e.g., absolute delta) but not others (e.g., slack) are not reselected in the second iteration of SI analysis.

When false, the union of sets of nets reselected by delta delay and slack based criteria is used. For a net to be reselected the following must be true: - The net is reselected by absolute delta delay OR - The net is reselected by relative delta delay OR - The net is reselected by setup OR hold slack OR borrowing OR

To determine the current value of this variable, type **printvar si_xtalk_reselect_delta_and_slack**.

.

SEE ALSO

si_xtalk_reselect_max_mode_slack(3)
si_xtalk_reselect_min_mode_slack(3)
si_xtalk_reselect_delta_delay(3)
si_xtalk_reselect_delta_delay_ratio(3)

si_xtalk_reselect_delta_delay

Specifies the threshold of net delay change caused by crosstalk analysis, above which IC Compiler reselects the net for subsequent delay calculations.

TYPE

float

DEFAULT

5

DESCRIPTION

This variable specifies a reselection threshold in terms of absolute delta delay. Nets that have at least one net arc with a crosstalk-annotated delta delay above this threshold are selected for the next iteration of PrimeTime-SI delay calculations. The units are the time units of the main library of the design.

This variable is one of a set of four variables that determine net reselection criteria. The other three variables are as follows:

si_xtalk_reselect_delta_delay_ratio
si_xtalk_reselect_max_mode_slack
si_xtalk_reselect_min_mode_slack

To determine the current value of this variable, type **printvar si_xtalk_reselect_delta_delay**.

SEE ALSO

si_xtalk_reselect_delta_delay_ratio(3)
si_xtalk_reselect_max_mode_slack(3)
si_xtalk_reselect_min_mode_slack(3)

si_xtalk_reselect_delta_delay_ratio

Specifies the threshold of the ratio of net delay change caused by crosstalk analysis to the total stage delay, above which IC-Compiler reselects a net for subsequent delay calculations.

TYPE

float

DEFAULT

0.95

DESCRIPTION

This variable specifies a reselection threshold in terms of the delta delay ratio. Nets that have at least one net arc with a crosstalk-annotated delta delay, where the ratio of the annotated delta to the stage delay is above this threshold, are selected for the next iteration of SI delay calculations.

If a net has multiple stage delays (because of a net fanout greater than one or multiple cell arcs), IC-Compiler considers the stage delta delay and stage delay that result in higher delta to stage delay ratio, thus making reselection conservative.

This variable is one of a set of four variables that determine net reselection criteria. The other three variables are as follows:

si_xtalk_reselect_delta_delay
si_xtalk_reselect_max_mode_slack
si_xtalk_reselect_min_mode_slack

To determine the current value of this variable, type **printvar si_xtalk_reselect_delta_delay_ratio**.

SEE ALSO

si_xtalk_reselect_delta_delay(3)
si_xtalk_reselect_max_mode_slack(3)
si_xtalk_reselect_min_mode_slack(3)

si_xtalk_reselect_max_mode_slack

Specifies the max mode pin slack threshold, below which IC-Compiler reselects a net for subsequent delay calculations.

TYPE

float

DEFAULT

0

DESCRIPTION

This variable specifies the pin slack threshold in the max mode. Nets that have at least one pin with a max mode slack below this threshold are selected for the next iteration of SI delay calculations. Max-mode pin slack is the slack of the worst max-mode (setup) path through the pin. The units are the time units of the main library of the design.

This variable is one of a set of four variables that determine net reselection criteria. The other three variables are as follows:

si_xtalk_reselect_delta_delay
si_xtalk_reselect_delta_delay_ratio
si_xtalk_reselect_min_mode_slack

All four variables are ignored if the variable **si_xtalk_reselect_critical_path** is true.

To determine the current value of this variable, type **printvar si_xtalk_reselect_max_mode_slack**.

SEE ALSO

si_xtalk_reselect_delta_delay(3)
si_xtalk_reselect_delta_delay_ratio(3)
si_xtalk_reselect_min_mode_slack(3)

si_xtalk_reselect_min_mode_slack

Specifies the min mode pin slack threshold, below which IC-Compiler reselects a net for subsequent delay calculations.

TYPE

float

DEFAULT

0

DESCRIPTION

This variable specifies the pin slack threshold in the min mode. Nets that have at least one pin with a min mode slack below this threshold are selected for the next iteration of SI delay calculations. Min-mode pin slack is the slack of the worst min-mode (hold) path through the pin. The units are the time units of the main library of the design.

This variable is one of a set of four variables that determine net reselection criteria. The other three variables are as follows:

si_xtalk_reselect_delta_delay
si_xtalk_reselect_delta_delay_ratio
si_xtalk_reselect_max_mode_slack

All four variables are ignored if the variable **si_xtalk_reselect_critical_path** is true.

To determine the current value of this variable, type **printvar si_xtalk_reselect_min_mode_slack**.

SEE ALSO

si_xtalk_reselect_delta_delay(3)
si_xtalk_reselect_delta_delay_ratio(3)
si_xtalk_reselect_max_mode_slack(3)

single_group_per_sheet

Specifies to the tool to put only one logic group on a sheet.

TYPE

Boolean

DEFAULT

false

GROUP

schematic_variables

DESCRIPTION

Specifies to the tool to put only one logic group on a sheet. Using this partitioning option set to *true* eliminates the possibility of more than one off-sheet connector with the same name being on a single sheet. The default value is *false*.

To determine the current value of this variable, type **printvar single_group_per_sheet**. For a list of all **schematic** variables and their current values, type **print_variable_group schematic**.

SEE ALSO

schematic_variables(3)

site_info_file

Contains the path to the site information file for licensing.

TYPE

string

DEFAULT

""

DESCRIPTION

Contains the path to the site information file for licensing. The default is the empty string.

site_row_attributes

Contains attributes related to site row.

DESCRIPTION

Contains attributes related to site row.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified site row. Specified with **list_attribute -class site_row -application**, the definition of attributes can be listed.

Site Row Attributes

cell_id
: Specifies Milkyway design ID in which a site row object is located.
This attribute is read-only.

name
: Specifies name of a site row object.
This attribute is read-only.

object_class
: Specifies object class name of a site row object, which is **site_row**.
This attribute is read-only.

object_id
: Specifies object ID in Milkyway design file.
This attribute is read-only.

bbox
: Specifies the bbox of a site row object.
This attribute is read-only.

bbox_ll
: Specifies the lower-left of a site row object.
This attribute is read-only.

bbox_llx
: Specifies x coordinate of the lower_left of the bbox a site row object.
This attribute is read-only.

bbox_lly
: Specifies y coordinate of the lower_left of the bbox a site row object.
This attribute is read-only.

bbox_ur
: Specifies the upper-right of the bbox of a site row object.
This attribute is read-only.

bbox_urx
: Specifies x coordinate of the upper-right of the bbox a site row object.

This attribute is read-only.

bbox_ury
: Specifies y coordinate of the upper-right of the bbox a site row object.
This attribute is read-only.

site_type
: Specifies the type of site being defined.
This attribute is read-only.

orientation
: Specifies the orientation of the sites. The value can be N, W, S, E, FN, FW, FS, FE.
This attribute is read-only.

direction
: Specifies the direction of the row. The value can be v, h, vertical and horizontal.
This attribute is read-only.

site_count
: Specifies the number of the sites in the row.
This attribute is read-only.

site_space
: Specifies the space for each site, from the lower left corner of the site to the lower left corner of the next site. The value is specified in microns.
This attribute is read-only.

origin
: Specifies the lower left corner of the row, regardless of orientation.
This attribute is read-only.

allowable_pattern
: Specifies the restricted orientations of placed cells on a specified row, which are based on what direction of row is, whether row is flipped and what direction of unit tile is.
This attribute is read-only.

row_type
: It is just a positive integer associated with site rows. Later on set_cell_row_type can be used to associate one particular cell with some particular rows of corresponding row_type.
This attribute can be read, set and removed.

is_ignored
: TRUE: ignore specified site row FALSE: The specified site row is not ignored
The attribute can be read and set.

SEE ALSO

get_attribute(2),
list_attribute(2),
report_attribute(2).

sort_outputs

Sorts output ports on the schematic by port name.

TYPE

string

DEFAULT

false

GROUP

schematic_variables

DESCRIPTION

Sorts output ports on the schematic by port name.

To determine the current value of this variable, use **printvar sort_outputs**. For a list of all **schematic** variables and their current values, use the **print_variable_group schematic** command.

suffix_variables

Variables that define the standard suffixes for files used by the design and dft compilers.

SYNTAX

```
string view_analyze_file_suffix = {"v", "vhd", "vhdl"}
list view_execute_script_suffix={".script", ".scr", ".dcs", ".dcsh", ".dc", ".con",
 ".wscr", ".rscr"}
list view_read_file_suffix = {"db", "sdb", "edif", "eqn", "fnc", "lsi", "mif", "NET
", "pla", "st",
                                  "tdl", "v", "vhd", "vhdl"}
list view_write_file_suffix = {"db", "sdb", "do", "edif", "eqn", "fnc", "lsi", "NET
", "neted",
                                    "pla", "st", "tdl", "v", "vhd", "vhdl", "xnf"}
```

DESCRIPTION

These variables define the standard suffixes for files used by the Synopsys design and dft compilers. File names with these suffixes are used by the Synopsys commands and **read** and **write** menus of the Design Analyzer viewer. Defaults are listed above, under Syntax.

For a list of these variables and their current values, type **print_variable_group suffix**. To view this manual page online, type **help suffix_variables**. To view an individual variable description, type **help *var***, where *var* is the variable name.

view_analyze_file_suffix
: A list of file extensions that specifies the files that are shown in the File/ Analyze dialog.

view_execute_script_suffix
: Used by the "Execute Script" option of the Setup menu and displays only files with these suffixes from directories you select in the option window.

view_read_file_suffix
: Used by the "Read" option of the Design Analyzer File menu and displays only files with these suffixes from directories selected in the option window.

view_write_file_suffix
: Used by the "Save As" option of the File menu and displays only files with these suffixes from directories selected in the option window.

SEE ALSO

design_analyzer (1).

suppress_errors

Specifies a list of error codes for which messages are to be suppressed during the current Design Analyzer/dc_shell session.

TYPE

list

DEFAULT

PWR-18 OPT-932 OPT-317 RCCALC-010 RCCALC-011

GROUP

system_variables

DESCRIPTION

Specifies a list of error codes for which messages are to be suppressed during the current Design Analyzer/dc_shell session. Default is set to no error message being suppressed.

To determine the current value of this variable, use **printvar suppress_errors**. For a list of all **system** variables and their current values, use the **print_variable_group system** command.

symbol_library

Specifies the symbol libraries to use during schematic generation.

TYPE

string

DEFAULT

your_library.sdb

GROUP

schematic_variables

DESCRIPTION

Specifies the symbol libraries to use during schematic generation. This variable is a list of symbol library names.

To determine the current value of this variable, use **printvar symbol_library**. For a list of all **schematic** variables and their current values, use the **print_variable_group schematic** command.

synlib_abort_wo_dw_license

Abort compile command if DesignWare license is required to compile the current design, but the DesignWare license is not available.

TYPE

Boolean

DEFAULT

false

GROUP

synlib_variables

DESCRIPTION

If the DesignWare license is required to compile the current design, but the DesignWare license is not available, abort the compile command.

The datapath generator flow requires DesignWare license. If the DesignWare license is not available, the free designware implementation will be used in the design. This could result in a suboptimal circuit.

If **synlib_abort_wo_dw_license** is true, compile command will abort with an error message. You can also set **synlib_wait_for_design_license** to wait for the licenses if all the licenses are checked out.

To determine the current value of this variable, type **printvar synlib_abort_wo_dw_license**. For a list of all **synlib** variables and their current values, type **print_variable_group synlib**.

SEE ALSO

synlib_wait_for_design_license(3)
synlib_variables(3)

synlib_dont_get_license

Specifies a list of synthetic library part licenses that the compiler does not automatically check out.

TYPE

list

DEFAULT

""

GROUP

synlib_variables

DESCRIPTION

Specifies a list of synthetic library part licenses that the compiler does not automatically check out. By default, the compiler checks out all synthetic library part licenses if there is a possibility that they will be used.

When there is only the possibility of a license being used, add the license to this list so that it will not be automatically checked out.

Exceptions exist in that licenses on this list are checked out, but only when a design is being read that requires these licenses or when you manually request them with the **get_license** command.

To determine what licenses are required to read a design, use the **report_design** command.

To determine the legality of the licenses on the list, use the **set_synlib_dont_get_license** command. This command checks each license on the list and issues a warning message if the license cannot be found in the key file. It then sets the **synlib_dont_get_license** variable.

To determine the current value of this variable, type **printvar synlib_dont_get_license**. For a list of all **synlib** variables and their current values, type **print_variable_group synlib**.

SEE ALSO

get_license(2)
report_design(2)
set_synlib_dont_get_license(2)
synlib_variables(3)

synlib_dwgen_smart_generation

Determines strategies used by the DesignWare arithmetic generators.

TYPE

Boolean

DEFAULT

true

GROUP

synlib_variables

DESCRIPTION

Determines whether the DesignWare arithmetic generators will use different strategies when building implementations of arithmetic parts or datapath blocks.

When set to *true* (the default), an *intelligent* generation strategy will be used to try to generate the best results. The user can control which strategies are available to the generators when smart generation is active using the command **set_dp_smartgen_options**.

When *false*, the DesignWare arithmetic generators will use a simple generation strategy.

To determine the value of this variable, type **printvar synlib_dwgen_smart_generation**. For a list of all **synlib** variables and their current values, type **print_variable_group synlib**.

This variable affects the datapath flow.

SEE ALSO

set_dp_smartgen_options(2)
report_dp_smartgen_options(2)
synlib_variables(3)

synlib_hiis_force_on_cells

Specifies a list of design cells on which the compiler is to force hierarchical incremental implementation selection (hiis).

TYPE

list

DEFAULT

""

GROUP

synlib_variables

DESCRIPTION

Specifies a list of design cells on which the compiler is to force hierarchical incremental implementation selection (hiis). Even when implementation is set on the subcomponent of a hierarchical synthetic component, the compiler ignores the implementation attribute on the subcomponent and performs hierarchical incremental implementation selection on the synthetic component.

The list defined by this variable contains valid cell names in the current design. A simple wildcard pattern can be supported by using the **find** command.

Hierarchical cell names are not supported. For example, to force hiis on cell U0/U1/MULT, do not place U0/U1/MULT in the **synlib_hiis_force_on_cells** list; put MULT in the list.

The compiler performs forced hiis only on instantiated DesignWare components that have single implementation. If you add to the **synlib_hiis_force_on_cells** list the cell name of an operator cell that is inferred in the design, the compiler does not perform forced hiis on the inferred cell. If there are multiple implementations available for a hierarchical DesignWare component, the compiler will not perform hierarchical incremental implementation selection on the DesignWare cell. To establish single implementation on such a cell, use the **set_implementation** command on the cell or use the **set_dont_use** command on all but one implementation.

To determine the current value of this variable, type **printvar synlib_hiis_force_on_cells**. For a list of all **synlib** variables and their current values, type **print_variable_group synlib**.

EXAMPLES

The following example places all cell names that begin with the letter *a* in the **synlib_hiis_force_on_cells** list:

```
prompt> synlib_hiis_force_on_cells = find(cell, "a*")
```

The following example places all cell names in the **synlib_hiis_force_on_cells** list:

```
prompt> synlib_hiis_force_on_cells = find(cell, "*")
```

SEE ALSO

find(2)
set_dont_use(2)
set_implementation(2)
synlib_variables(3)

synlib_iis_use_netlist

Allows netlists of the DesignWare parts to be used, instead of timing models, for cost comparison during the Incremental Implementation Selection step.

TYPE

Boolean

DEFAULT

false

GROUP

synlib_variables

DESCRIPTION

Allows netlists of the DesignWare parts to be used as opposed to timing models for cost comparison during the Incremental Implementation Selection step. Timing models are created without considering the design context; netlists are mapped in the context of the design.

When set to *true*, netlist of the DesignWare architectures are used to calculate the timing and area information.

When *false*, timing models of the DesignWare architectures are used.

SEE ALSO

synlib_variables(3)
compile(2)

synlib_variables

SYNTAX

string **cache_dir_chmod_octal** = "777"
string **cache_file_chmod_octal** = "666"
list **cache_read** = {"/remote/release/v3.1/libraries/syn","~"}
Boolean **cache_read_info** = "false"
string **cache_write** = "~"
Boolean **cache_write_info** = "false"
Boolean **mgi_scratch_directory** "designware_generator"
Boolean **synlib_disable_limited_licenses** = "true"
list **synlib_dont_get_license** = {}
Boolean **synlib_evaluation_mode** = "false"
string **synlib_model_map_effort** = "low"
Boolean **synlib_optimize_non_cache_elements** = "true"
Boolean **synlib_prefer_ultra_license** = "false"
string **synlib_sequential_module** = "default"
list **synlib_wait_for_design_license** = {}
list **synthetic_library** = {}
Boolean **synlib_dwgen_smart_generation** = "true"
Boolean **compile_report_dp** = "false"

DESCRIPTION

Cache files are created with their mode bits set to the value of **cache_file_chmod_octal**. These variables directly affect the **cache_ls** command.

For a list of these variables and their current values, type **print_variable_group synlib**. To view this manual page online, type **help synlib_variables**. To view an individual variable description, type **help *var***, where *var* is the variable name.

cache_dir_chmod_octal
: A string that is translated to an octal number and sets the mode for the cache directory. There are separate variables for directories and files to allow the sticky bit to be set.

cache_file_chmod_octal
: A string that is translated to an octal number and sets the mode for the cache file.

cache_read
: A list of directories. Each directory can contain a cache that will be read from whenever a cache entry is needed.

cache_read_info
: When *true*, an informational message will be printed each time a cache element is read. The default is *false*.

cache_write
: The name of the directory where optimized and unoptimized synlib parts are written, if they are not already in the cache.

cache_write_info

When *true*, an informational message will be printed each time a cache element is written. The default is *false*.

mgi_scratch_directory

Specifies a directory in which to store the intermediate files created by external generator(s). The default is designware_generator in the current directory.

synlib_disable_limited_licenses

When *true* (the default), limited licenses for synthetic library parts are not considered. No limited licenses will be checked out automatically. Limited licenses will not enable synthetic library parts.

synlib_dont_get_license

Specifies a list of synthetic library part licenses that are not automatically checked out. By default, all synthetic library part licenses are automatically checked out if there is a possibility that they will be used.

synlib_evaluation_mode

When no Designware-Basic or Designware-FPGA-Basic keys are available, you can still evaluate synthetic library parts by setting this variable to "true". When this variable is set, you will be authorized for almost the same capabilities that Designware-Basic would provide. The only difference is that all DesignWare parts and the level of hierarchy that contains them will automatically be given a limited license. This means that it will be impossible to write out any DesignWare part (including standard parts) or any design that contains a DesignWare part. You will still be able to print reports that show the timing (report_timing) and the implementation selections made (report_resource).

synlib_model_map_effort

Determines the **map_effort** used during the modeling of synthetic library parts. Allowed values are *low* (the default value), *medium*, and *high*.

synlib_optimize_non_cache_elements

When *true* (the default value), creates and optimizes part descriptions of non-cached models (that is, models that cannot be retrieved from the cache). When *false*, the non-cached part description is created and used in an unoptimized form.

synlib_sequential_module

Controls the amount of processing to be done during resource sharing and implementation selection by **compile** on synthetic library modules that have implementations with sequential elements. Allowed values are *default* (the default), *iis_processing*, *one_implementation_choice*, and *multiple_implementation_choices*. These values are described in the **synlib_sequential_module** manual page.

synlib_prefer_ultra_license

When *true*, any use of Foundation library parts sets the Design Compiler ultra optimization mode and checks out the DesignWare-Foundation-Ultra license instead of the DesignWare-Foundation license. When *false* (the default), the DesignWare-Foundation license is used when Design Compiler is not in the ultra optimization mode.

synlib_wait_for_design_licenses
  Specifies a list of authorized synthetic library part licenses to be waited for. By default, Design Compiler terminates the command when none of the required design licenses are available. When this variable is *true*, Design Compiler waits for the licenses on the list to become avaiable and continues with the process instead of terminating the command.

synlib_dwgen_smart_generation
  When *true*, DesignWare arithmetic generators try to use a *smart* strategy to build implementations of arithmetic parts or datapath blocks. The exact strategy used depends on the variable **synlib_dwgen_smart_generation_options**. When *false*, a simple strategy is used.

synlib_dwgen_smart_generation_options
  The value of this variable controls the different *smart* generation strategies in the DesignWare arithmetic generators. Specific values are described in the **synlib_dwgen_smart_generation_options.3** manual page.

synthetic_library
  A list of synthetic libraries to use when compiling.

compile_report_dp
  Generates a detailed datapath extraction report during compile. This feature is available only in DC Ultra flow.

SEE ALSO

cache_ls (2).

synlib_wait_for_design_license

Specifies a list of authorized synthetic library licenses that Design Compiler is to wait for.

TYPE

list

DEFAULT

""

GROUP

synlib_variables

DESCRIPTION

Specifies a list of authorized synthetic library licenses that Design Compiler is to wait for.

By default, Design Compiler checks for all design licenses and checks out one that is available. If none of the licenses are available, Design Compiler terminates the command that requires the license. You can override this default behavior by setting a list of licenses as the value of **synlib_wait_for_design_license**. Then, if no licenses are available, Design Compiler does not terminate the command that requires the license, but waits for the listed licenses to become available.

To determine the current value of this variable, type **printvar synlib_wait_for_design_license**. For a list of all **synlib** variables and their current values, type **print_variable_group synlib**.

SEE ALSO

get_license(2)
synlib_variables(3)

syntax_check_status

Reports whether the **syntax_check** mode is enabled.

TYPE

Boolean

DEFAULT

false

GROUP

system_variables

DESCRIPTION

Reports whether the **syntax_check** mode is enabled. This is a read-only variable and the user cannot set it. This variable is one of a pair of status variables, **syntax_check_status** and **context_check_status**, whose values are set by the syntax checker and not by the user. You examine these variables to determine the status of the **syntax_check** or **context_check** mode of the syntax checker.

A value of *true* indicates that the mode is enabled. A value of *false* (the default) indicates that the mode is disabled. For example, a value of **syntax_check_status = true** indicates that the **syntax_check** mode is enabled. The two modes cannot be enabled simultaneously. These two status variables, **syntax_check_status** and **context_check_status**, allow you to determine whether one mode is enabled before you attempt to enable the other mode.

You enable or disable the **syntax_check** and **context_check** modes by executing the commands **syntax_check** or **context_check**. Alternatively, you can invoke **dc_shell**, **design_compiler**, and **dp_shell** in either the **context_check** or **syntax_check** mode by using the **-syntax_check** or **-context_check** options when you invoke the tool. For more information, refer to the manual pages of these commands or to the *Design Compiler Reference Manual*.

To determine the value of this variable, type **printvar syntax_check_status** or **echo syntax_check_status**. For a list of **system** variables and their current values, type **print_variable_group system**.

SEE ALSO

dc_shell(1)
design_analyzer(1)
dp_shell(1)
context_check(2)
syntax_check(2)
context_check_status(3)
system_variables(3)

synthetic_library

Specifies a list of synthetic libraries to use when compiling.

TYPE

list

DEFAULT

""

GROUP

system_variables, synlib_variables

DESCRIPTION

Specifies a list of synthetic libraries to use when compiling. Default is {}.

The **synthetic_library** variable works much like the **target_library** variable does for technology libraries. This variable can be set to be a list of zero or more sldb files that you wish to use in the **compile** or **replace_synthetic** commands. When synthetic operators or modules are processed in compile, the operators, bindings, modules, and implementations of the specified library or libraries are used. Synthetic libraries are processed in order. So, if two modules in different libraries have the same name, the module in the first listed library is used.

As with target technology libraries, it is sometimes necessary to include your synthetic library as part of the link_library set. This is especially important when you instantiate synthetic modules.

Because HDL files automatically insert synthetic operators in a netlist, it is important to have a synthetic library defined that supports these operators. For this reason the standard Synopsys library **standard.sldb** is automatically inserted as the first entry in the **synthetic_library** variable list. Then if a synthetic library is specified to be an empty list, the insertion of **standard.sldb** provides default definitions.

There is no way to disable the standard synthetic library, but you can disable individual modules or implementations by using the **dont_use** command. To replace a particular implementation, disable it with **dont_use**, and use a replacement from your own synthetic library.

To determine the current value of this variable, use **printvar synthetic_library**. To see a list of all **system** variables and their current values, use the **print_variable_group system** command.

SEE ALSO

compile(2)

```
replace_synthetic(2)
target_library(3)
```

system_variables

SYNTAX

Boolean **auto_link_disable** = "false"
string **auto_link_options** = "-all"
Boolean **change_names_dont_change_bus_members** = "false"
string **command_log_file** = "./command.log"
string **company** = ""
string **compatibility_version** = ""
Boolean **context_check_status** = true/false
string **current_design** = "<<undefined>>"
string **current_instance** = "<<undefined>>"
(any type) **dc_shell_status** = 1
string **default_name_rules** = ""
string **designer** = ""
Boolean **echo_include_commands** = "true"
Boolean **enable_page_mode** = "true"
string **exit_delete_filename_log_file** = "true"
string **filename_log_file** = "filenames.log"
Boolean **find_converts_name_lists** = "false"
string **link_force_case** = "check_reference"
list **link_library** = {your_library.db}
list **search_path** = {. synopsys_root + "/libraries"}
list **suppress_errors** = {}
Boolean **syntax_check_status** = true/false
list **synthetic_library** = {}
list **target_library** = {your_library.db}
string **uniquify_naming_style** = "%s_%d"
Boolean **verbose_messages** = "false"

DESCRIPTION

These variables directly affect Synopsys Design Compiler and DFT Compiler commands.

For a list of these variables and their current values, type **print_variable_group system**. To view this manual page online, type **help system_variables**. To view an individual variable description, type **help *var***, where *var* is the variable name.

auto_link_disable

when *true*, disables automatic linking of the design. Several Design Compiler and DFT Compiler commands (including **create_schematic** and **compile**) issue the **link** command automatically. During the execution of scripts containing thousands of **set_load** and **set_resistance** commands, the overhead of automatic linking can be significant. This variable can be used in this situation to speed up the execution of such scripts. The default is *false*.

auto_link_options

Specifies the **link** command options to be used when **link** is invoked automatically by various Design Compiler and DFT Compiler commands (for example, **create_schematic** and compile). The default is **-all**. To find the available options, refer to the **link** command manual page.

change_names_dont_change_bus_members

This variable is for the **change_names** command. It affects bus members only of bussed ports or nets. When *false* (the default), **change_names** gives bus members the base name from their owning bus. For example, if BUS A has range 0 to 1 with the first element NET1 and the second element NET2, **change_names** changes NET1 to A[0] and NET2 to A[1]. When this variable is set to *true*, **change_names** does not change the names of bus members, so that NET1 and NET2 remain unchanged.

command_log_file

Names the file to which a log of the initial values of variables and commands executed is to be written. If the value is an empty string, a command log file is not created.

company

Names the company where Synopsys software is installed. The company name is displayed on the schematics.

compatibility_version

The name of the Synopsys software version to which the default behavior of the system should be set. This provides compatibility for script command files written in previous software versions. The scripts actually are run on the current version of the software, so results are usually better. However, the script performs the same default actions as it did on the specified software version.

context_check_status

One of a pair of status variables, **syntax_check_status** and **context_check_status**, whose values are set by the Syntax Checker and not by the user. A value of *true* indicates that the context_check mode is currently enabled; a value of *false* indicates that the mode is disabled.

current_design

The name of the design to be worked on. This variable is used by most of the Synopsys commands. Until a design is read into the system as the **current_design**, the value of **current_design** is "<<undefined>>". When one or more designs are read into the system, any of them can be made current by assigning the design name to this variable.

current_instance

The name of the instance to be worked on. Until a design is read into the system as the **current_design**, the value of the **current_instance** is "<<undefined>>". When a design is read into the system at the top hierarchical level, the value of **current_instance** is the top design name followed by a slash.

dc_shell_status

Contains the return value of the previously executed command. Therefore, the variable can be of any type. Most commands return the integer 0 to indicate failure or 1 to indicate successful execution. This variable is most often used as the conditional expression of an **if** or **while** command.

default_name_rules

Contains the name of a name_rules file to be used as a default by **change_names** if a name_rules file is not specified using the **-rules** *name_rules* option.

change_names changes the names of ports, cells, and nets to conform to the rules contained in the file specified by **default_name_rules**. For information on the format and creation of a name_rules file, refer to the **change_names** manual page.

designer

The name of the current user. This name is displayed on the schematics.

echo_include_commands

When *true* (the default value), the **include** command prints the contents of files that it executes. When *false*, contents are not printed.

enable_page_mode

When *true* (the default value), long reports are displayed one page at a time (similar to the UNIX **more** command). Commands affected by this variable include the **list**, **help**, and **report** commands.

exit_delete_filename_log_file

When *true* (the default value), causes the file specified by the variable **filename_log_file** to be deleted after **design_analyzer** or **dc_shell** exits normally. Set **exit_delete_filename_log_file** to *false* if you want the file to be retained.

filename_log_file

Specifies the name of the filename log file to be used in case a fatal error occurs during the execution of **design_analyzer** or **dc_shell**. The file specified by **filename_log_file** will contain all the filenames read in by **design_analyzer** or **dc_shell**, including db, script, verilog, vhdl or include files for one invocation of the program. If there is a fatal error, you can easily identify the data files needed to reproduce the fatal error. If this variable is not specified, the default filename **files.log** will be used. This file will be deleted if the program exits normally, unless **exit_delete_filename_log_file** is set to *false*.

find_converts_name_lists

When *true*, directs the **find** command to convert the *name_list* string to a list of strings before searching for design objects. In addition, when this variable is *true*, all commands that use the implicit **find** will convert appropriate strings to lists of strings before searching for objects. When the variable is *false* (the default value), strings are not converted to lists of strings.

link_force_case

Controls the case-sensitive or case-insensitive behaviour of the **link** command. Values are *case_sensitive*, *case_insensitive*, or *check_reference*. The default is *check_reference*, which causes **link** to check and enforce the case sensitivity of the input format that created the reference.

link_library

Specifies the list of design files and libraries used during linking. The **link** command looks at those files and tries to resolve references in the order of specified files. If file names do not include directory names, files are searched for in the directories in **search_path**. The default is {your_library.db}. You should change this to reflect your library name.

search_path

Specifies directories searched by the Design Compiler and DFT Compiler for files specified without directory names. This includes looking for technology and symbol libraries, design files, and so forth. This variable is a list of directory names and is usually set to a central library directory.

suppress_errors

Specifies a list of error codes for which messages are to be suppressed during the current Design Analyzer/**dc_shell** session. The default is set to no error message being suppressed.

syntax_check_status

One of a pair of status variables, **syntax_check_status** and **context_check_status**, whose values are set by the Syntax Checker and not by the user. A value of *true* indicates that the syntax_check mode is currently enabled; a value of *false* indicates that the mode is disabled.

synthetic_library

Specifies a list of synthetic libraries to use when compiling. The default is {}.

target_library

Specifies the list of technology libraries of components used when compiling a design. The default is {your_library.db}. You should change this to reflect your library name.

uniquify_naming_style

Specifies the naming convention used by **uniquify**. The variable string must contain only one %s (percent s) and %d (percent d) character sequence. To use a percent sign in the design name, two are needed in the string (%%).

verbose_messages

When *true*, causes more explicit system messages to be displayed during the current Design Analyzer/**dc_shell** session. The default is *false*.

SEE ALSO

dc_shell (1), **change_names** (2), **compile** (2), **context_check** (2), **create_schematic** (2), **current_instance** (2), **current_design** (2), **find** (2), **help** (2), **if** (2), **include** (2), **link** (2), **list** (2), **report** (2), **syntax_check** (2), **uniquify** (2), **while** (2).

systemcout_debug_mode

TYPE

Boolean

DEFAULT

false

GROUP

scc_variables

DESCRIPTION

This variable is to be used only when the **systemcout_levelize** variable is set to *true*, to generate debug information.

When **systemcout_debug_mode** is *true*, and if **systemcout_levelize** is also *true*, Behavioral Compiler, when generating levelized RTL SystemC code for simulation, inserts additional code to write out debugging information during the RTL simulation run. When **systemcout_debug_mode** is *false* (the default), Behavioral Compiler does not insert this additional code.

The debugging information written out during simulation includes the following:

1. A SystemC process tracing the execution of the state machine in the RTL design. This process contains variables that can be monitored within the debugger/simulator, to obtain the following information:

   o The current state of the state machine
   o Which loop in the behavior is currently being executed
   o The number of clock cycles used so far in the current loop

2. Assertion warnings about registers that are being set to unknown values.

3. Assertion warnings about multiplexers whose control inputs have invalid values.

4. A trace of all memory reads and writes, which includes the address and data being read or written. For memory locations being set to unknown values, a warning is issued.

5. A trace of all I/O operations, which includes the design name, the port name, the scheduled control step, the value read or written, and the simulation cycle.

6. When an input port has had automatic register allocation disabled using the **bc_dont_register_input_port** command, a value on that port is supposed to remain constant. Therefore, if the value on such a port changes, a warning message is issued.

Note: To enable features 5 and 6, you must set this variable to *true* before invoking the **schedule** command.

EXAMPLES

The following example shows a section of the process tracing the state machine execution.

```
// Calculate the state of the FSM
 state_bits[5] =   n219 ;
 state_bits[4] =   n220 ;
 state_bits[3] =   n221 ;
 state_bits[2] =   n222 ;
 state_bits[1] =   n223 ;
 state_bits[0] =   n224 ;

 if ( state_bits == "100000"  ) {
   state = "s_0_0";
 }
     .
     .
     .
 } else {

   cout << "FSM 'entry_ctl_state' is in an unknown state"\
   << " at simulation time : " << sc_time_stamp() << endl;
   state = "UNKNOWN";
 }
```

Use *state* to monitor the current state of the state machine, *current_loop* to monitor the current loop being executed, and the variable named after the current loop (for example, **reset_loop**) to monitor the clock cycles used in the loop.

The following example shows an assertion warning about a register being set to an unknown value.

```
Register 'out_value1_reg_Q' has been set to X's
```

The following example shows a warning about invalid multiplexer settings.

```
Control lines of selector s_out_valid_reg_controls have non 0/1 values
```

The following example shows a report of a memory write.

```
 r1_Q is writing to memory RAM_A[seq_cell_5_ADDR]
 seq_cell_5_ADDR = 0
```

```
    RAM_A[seq_cell_5_ADDR] = 506  r1_Q = 506 at simulation time: 190
```

The following is an example of I/O tracing.

```
if1:  (cycle: 75)  READin_valid@cstep0x11
if1:  (cycle: 76)  READin_value2@cstep0x356
if1:  (cycle: 76)  READcontrol_in@cstep0x31
if1:  (cycle: 77)  WRITEout_value1@cstep0x428
if1:  (cycle: 77)  WRITEout_valid@cstep0x40
```

The following is an example of the warning on the value change on port in_value1, for which automatic register allocation has been disabled using the **bc_dont_register_input_port** command.

```
Warning: The value on port in_value1 (w/
 bc_dont_register_input_port set) has changed.
```

SEE ALSO

systemcout_levelize(3)

systemcout_levelize

Levelizes and flattens the netlist and replaces standard DesignWare operations with simulatable SystemC, before writing out the netlist, during **write -f systemc** command activity.

TYPE

string

DEFAULT

true

GROUP

scc_variables

DESCRIPTION

Levelizes and flattens the netlist (when true) and replaces standard DesignWare operations with simulatable SystemC, before writing out the netlist, during **write -f systemc** command activity.

In a levelized netlist, combinational gates are written before any combinational gates that they feed. Thus the netlist can be written out as SC_METHOD processes sensitive to the clock, generating fewer simulation events to be processed by the simulator. Because the processing of simulation events is very time-consuming, reducing the number of these events results in faster simulation time. Note that the hierarchical levels in the design are removed.

The command **write -f systemc** does not produce a synthesizable (nonlevelized) output written in the SystemC language. To generate synthesizable RTL level output, use the **write -f verilog** or **write -f vhdl** commands.

NOTE: Setting this variable to true has no effect if the design has not been run through SystemC/Behavioral Compiler, or if the design has been mapped. Also, designs that contain combinational feedback cannot be levelized.

To determine the current value of this variable, type **printvar systemcout_levelize**.

SEE ALSO

write(2)
systemcout_debug_mode(3)

target_library

Specifies the list of technology libraries of components to be used when compiling a design.

TYPE

list

DEFAULT

your_library.db

GROUP

system_variables

DESCRIPTION

Specifies the list of technology libraries of components to be used when compiling a design. The default is {your_library.db}. Change this value to reflect your library name.

To determine the current value of this variable, use **printvar target_library**. For a list of all **system** variables and their current values, use the **print_variable_group system** command.

SEE ALSO

compile(2)
system_variables(3)

template_naming_style

Generates automatically a unique name when a module is built.

TYPE

string

DEFAULT

GROUP

hdl_variables

DESCRIPTION

Generates automatically a unique name when a module is built. This variable is one of three string variables that determine the naming conventions for parameterized modules (templates) built into a design through the **elaborate** command or automatically instantiated from an HDL file. The unique name automatically generated uses the module name, parameter names, and parameter values.

The **template_naming_style** variable determines what character or characters appear between the design name and the parameter name. The string value must contain %s, which stands for the name of the original design, and %p, which stands for the name and value of the parameter or parameters. You can optionally include any ASCII character or characters, or none, between %s and %p. For example, for a design named *DesignName* that has a parameter *parm1*, the default %s_%p causes the name **DesignName_parm1** to be generated.

Further, %s$%p, %s_*_%p, and %s%p would generate respectively the names **DesignName$parm1**, **DesignName_*_parm1**, and **DesignNameparm1**.

If a design has a noninteger parameter (or if **template_naming_style = ""**), the following definitions are locked down for these variables:

```
template_naming_style = %s_%p
template_parameter_style = %d
template_separator_style = _
```

To determine the current value of this variable, type **printvar template_naming_style**. For a list of all **hdl** variables and their current values, type **print_variable_group hdl**.

SEE ALSO

elaborate(2)
hdl_variables(3)
template_parameter_style(3)
template_separator_style(3)

template_parameter_style

Generates automatically a unique name when a module is built.

TYPE

string

DEFAULT

GROUP

hdl_variables

DESCRIPTION

Generates automatically a unique name when a module is built. This variable is one of three string variables that determine the naming conventions for parameterized modules (templates) built into a design through the **elaborate** command or automatically instantiated from an HDL file. The unique name automatically generated uses the module name, parameter names, and parameter values.

The **template_parameter_style** variable determines what character or characters appear between the parameter name and its value.

The string must contain %s, which stands for the name of the parameter, and %d, which stands for the value of the parameter. You can optionally include any ASCII character or characters, or none, between %s and %d. For example, for a parameter named *parm* that has a value of *1*, the default %s%d causes the name **parm1** to be generated.

Other examples:

template_naming_style = "%s$%p"
template_parameter_style = %s_%d   results in **DesignName$parm_1**

template_naming_style = "%s_%p"
template_parameter_style = %s@%d   results in **DesignName_parm@1**

If a design has a noninteger parameter (or if **template_naming_style = ""**), the following definitions are locked down for these variables:

template_naming_style = %s_%p
template_parameter_style = %d
template_separator_style = _

To determine the current value of this variable,
type **printvar template_parameter_style**.
For a list of all **hdl** variables and their current values,
type **print_variable_group hdl**.

SEE ALSO

```
elaborate(2)
hdl_variables(3)
template_naming_style(3)
template_separator_style(3)
```

template_separator_style

Generates automatically a unique name when a module is built.

TYPE

string

DEFAULT

_

GROUP

hdl_variables

DESCRIPTION

Generates automatically a unique name when a module is built. This variable is one of three string variables that determine the naming conventions for parameterized modules (templates) built into a design through the **elaborate** command or automatically instantiated from an HDL file. The unique name automatically generated uses the module name, parameter names, and parameter values.

The **template_separator_style** variable determines what character or characters appear between parameter names for templates that have more than one parameter. You can designate any ASCII character or characters, or none. The default value is an underscore (_).

For example, for a design called *DesignName* that has parameters named *parm1*, *parm2*, and *parm3*, if **template_naming_style = "%s_%p"** (the default value), and **template_separator_style = "_"**, the name **DesignName_parm1_parm2_parm3** is generated.

Other examples:

```
template_naming_style = "%s$%p"
template_separator_style = "_"     results in DesignName$parm1_parm2_parm3

template_naming_style = "%s#%p"
template_separator_style = "/"     results in DesignName#parm1/parm2/parm3
```

If a design has a noninteger parameter (or if **template_naming_style = ""**), the following definitions are locked down for these variables:

```
        template_naming_style = %s_%p
        template_parameter_style = %d
        template_separator_style = _
```

To determine the current value of this variable, type **printvar template_separator_style**.

For a list of all **hdl** variables and their current values,
type **print_variable_group hdl**.

SEE ALSO

elaborate(2)
hdl_variables(3)
template_naming_style(3)
template_parameter_style(3)

terminal_attributes

Contains attributes related to terminal.

DESCRIPTION

Contains attributes related to terminal.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified object. Specified with **list_attribute -class terminal -application**, the definition of attributes can be listed.

Terminal Attributes

access_direction

Specifies allowable access directions for a terminal object.
The data type of **access_direction** is string.
Its valid values are:

- **right**
- **left**
- **up**
- **down**
- **unknown**

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox

Specifies the bounding-box of a terminal. The **bbox** is represented by a **rectangle**.
The format of a *rectangle* specification is {{llx lly} {urx ury}}, which specifies the lower-left and upper-right corners of the rectangle.
The data type of **bbox** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_ll

Specifies the lower-left corner of the bounding-box of a terminal.
The **bbox_ll** is represented by a **point**. The format of a *point* specification is {x y}.

You can get the **attr_name** of a terminal, by accessing the first element of its **bbox**.
The data type of **bbox_ll** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_llx
Specifies x coordinate of the lower-left corner of the bounding-box of a terminal.
The data type of **bbox_llx** is double.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_lly
Specifies y coordinate of the lower-left corner of the bounding-box of a terminal.
The data type of **bbox_lly** is double.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_ur
Specifies the upper-right corner of the bounding-box of a terminal.
The fbbox_ur is represented by a **point**. The format of a *point* specification is {x y}.
You can get the **bbox_ur** of a terminal, by accessing the second element of its **bbox**.
The data type of **bbox_ur** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_urx
Specifies x coordinate of the upper-right corner of the bounding-box of a terminal.
The data type of **bbox_urx** is double.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_ury
Specifies y coordinate of the upper-right corner of the bounding-box of a terminal.
The data type of **bbox_ury** is double.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

cell_id
Specifies Milkyway design ID in which a terminal object is located.
The data type of **cell_id** is integer.
This attribute is read-only.

constrained_status
Specifies constrained status of a terminal object.
The data type of **constrained_status** is string.
Its valid values are:

- **manual_created**
- **side**
- **location**
- **order**
- **unknown**

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

direction
: Specifies direction of a terminal object.
The data type of **direction** is string.
Its valid values are:

- **in**
- **out**
- **inout**
- **tristate**
- **unknown**
- **input**
- **output**

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

eeq_class
: Specifies electrically equivalent class of a terminal object.
The data type of **eeq_class** is integer.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

is_fixed

Specifies whether a terminal object is marked as fixed.
The data type of **is_fixed** is boolean.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

layer

Specifies layer name of a terminal object.
The data type of **layer** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

must_join_class

Specifies must-join class of a terminal.
The data type of **must_join_class** is integer.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

name

Specifies name of a terminal object.
The data type of **name** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

number_of_points

Specifies the number of points to illustrate the boundary of a terminal object.
The data type of **number_of_points** is integer.
You can refer to the attribute **points**. The list length of **points** is the value of **number_of_points**.
This attribute is read-only.

object_class

Specifies object class name of a terminal, which is **terminal**.
The data type of **object_class** is string.
This attribute is read-only.

object_id

Specifies object ID in Milkyway design file.
The data type of **object_id** is integer.
This attribute is read-only.

owner

Specifies Milkyway design file name in which a terminal is located.
The data type of **owner** is string.
This attribute is read-only.

owner_port

Specifies port name which a terminal object is associated with.
The data type of **owner_port** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

points

Specifies points of the boundary of a terminal. A terminal can be a rectangle,

a rectilinear polygon, or multiple rectangles.
When a terminal is either a rectangle or a rectilinear polygon, its **points** is represented by a list of points. The last element of the list is the same as the first element.
When a terminal consists of multiple rectangles, its **points** is represented by a list of points of rectangles. Every five points represent one rectangle.
The data type of **points** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

side_status

Specifies constrained side name.
The data type of **side_status** is string.
Its valid values are:

- **left**
- **right**
- **bottom**
- **top**
- **unknown**

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

status

Specifes PR status of a terminal object.
The data type of **status** is string.
Its valid values are:

- **fixed**
- **placed**
- **cover**
- **unplaced**

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

SEE ALSO

get_attribute(2),
list_attribute(2),
report_attribute(2),
set_attribute(2).

test_allow_clock_reconvergence

Allows reconvergent nets to originate from the same top-level clock port.

TYPE

string

DEFAULT

true

GROUP

test_variables

DESCRIPTION

Allows reconvergent nets to originate from the same top-level clock port. When *true* (the default), this variable instructs the **check_test** command to allow multiple edges to reconverge at a net and not to generate an X, unless the edges actually conflict.

Notice that **check_test** or **test_drc** still generates an X when multiple edges reconverge at a gate, as well as for all reconvergent edges that originate from different top-level clock ports.

When *false*, **check_test** or **test_drc** automatically assigns an X (unknown) whenever multiple edges meet at a gate or net. Because the skew between edges in the circuit is unknown, **check_test** or **test_drc** is pessimistic and does not consider delays when checking clocks.

To determine the current value of this variable, type **printvar test_allow_clock_reconvergence**. For a list of all **test** variables and their current values, type **print_variable_group test**.

SEE ALSO

test_bsd_allow_tolerable_violations

Allows the **optimize_bsd** command to replace observe_and_control BSR cells with observe_only cells or remove BSR cells during timing-driven or area-driven optimization.

TYPE

Boolean

DEFAULT

false

GROUP

bsd_variables

DESCRIPTION

Allows the **optimize_bsd** command to replace observe_and_control boundary-scan register (BSR) cells with observe_only cells, or to remove BSR cells, during timing-driven or area-driven optimization. This can give rise to tolerable violation during execution of the **check_bsd** command. When *false* (the default), cells cannot be replaced.

To determine the current value of this variable, type **printvar test_bsd_allow_tolerable_violations**. For a list of bsd variables and their current values, type **print_variable_group bsd**.

SEE ALSO

check_bsd(2)
test_bsd_control_cell_drive_limit(3)
test_bsd_optimize_control_cell(3)

test_bsd_control_cell_drive_limit

Specifies the number of cells a single BSR control cell can drive while optimizing control cell allocation during **optimize_bsd** command activity

TYPE

integer

DEFAULT

0

GROUP

bsd_variables

DESCRIPTION

Specifies the number of cells a single boundary-scan register (BSR) control cell can drive while optimizing control cell allocation during **optimize_bsd** command activity

For optimization to occur, set the **test_bsd_optimize_control_cell** variable to *true*. If area constraints are not violated, the optimization does not take place. To assign area constraints to the current design, use the **set_max_area** command.

To determine the current value of this variable, type **printvar test_bsd_control_cell_drive_limit**. For a list of **bsd** variables and their current values, type **print_variable_group bsd**.

SEE ALSO

set_max_area(2)
test_bsd_allow_tolerable_violations(3)
test_bsd_optimize_control_cell(3)

test_bsd_default_bidir_delay

Defines the default switching time of bidirectional ports in a tester cycle for bsd applications.

TYPE

positive nonzero integer

DEFAULT

0.0

GROUP

test_variables

DESCRIPTION

Defines the default switching time of bidirectional ports in a tester cycle. The value is a positive real number in nanoseconds. This value is used when a test protocol is generated for ATPG.

For ATPG, **test_bsd_default_bidir_delay** defines for the design under test the following:

> The default time at which values are applied (driven) to the bidirectional ports in input mode during parallel measure cycle.
>
> The time at which bidirectional ports are released (undriven) during capture cycle.
>
> This variable affects only bsd applications.

The value of **test_bsd_default_bidir_delay** must be less than the output strobe time and greater than the capture clock active edge value.

The value of this variable affects the outcome of the **check_bsd** command and the test program produced by the **create_bsd_patterns** or **write_test** commands. The **check_bsd** command uses the value of this variable to check a design against the design rules of a bsd methodology.

Changing the value of **test_bsd_default_bidir_delay** modifies the content of the inferred or created test protocol generated by **check_bsd**. To generate the test program, **create_bsd_patterns** or **write_test** uses the value of **test_bsd_default_bidir_delay** in the test protocol attached to the vector database (.vdb) file.

To determine the current value of this variable, type **printvar test_bsd_default_bidir_delay**. For a list of all **test** variables and their current values, type **print_variable_group test**.

SEE ALSO

```
check_bsd(2)
write_test(2)
port_attributes(3)
test_bsd_default_delay(3)
test_bsd_default_strobe(3)
test_bsd_default_strobe_width(3)
```

test_bsd_default_delay

Defines the default time in a tester cycle to apply values to input ports for bsd applications.

TYPE

positive nonzero integer

DEFAULT

0.0

GROUP

test_variables

DESCRIPTION

Defines the default time in a tester cycle to apply values to input ports. The value is a positive real number in nanoseconds. The default value is 5. This variable is used to generate a protocol file for ATPG.

For ATPG, **test_bsd_default_delay** defines for the design under test the default time at which values are applied (driven) to the primary inputs for bsd applications. The value of **test_bsd_default_delay** must be less than the output strobe time, and less than the capture clock edge value.

The value of this variable affects the outcome of the **check_bsd** command and the test program produced by the **create_bsd_patterns** or **write_test** commands. The **check_bsd** command uses the value of this variable to check a design against the design rules of a bsd methodology. Changing the value of **test_bsd_default_delay** modifies the content of the inferred or created bsd protocol generated by **check_bsd**.

To generate the test program, **create_bsd_patterns** or **write_test** uses the value of **test_bsd_default_delay** in the test protocol attached to the vector data base (.vdb) file.

To determine the current value of this variable, type **printvar test_bsd_default_delay**. For a list of all **test** variables and their current values, type **print_variable_group test**.

SEE ALSO

check_bsd(2)
write_test(2)
port_attributes(3)
test_bsd_default_bidir_delay(3)
test_bsd_default_strobe(3)
test_bsd_default_strobe_width(3)

test_bsd_default_strobe

Defines the default strobe time in a test cycle for output ports and bidirectional ports in output mode for bsd applications.

TYPE

positive nonzero integer

DEFAULT

95.0

GROUP

test_variables

DESCRIPTION

Defines the default strobe time in a test cycle for output ports and bidirectional ports in output mode for bsd applications. The value is a positive real number in nanoseconds. This value is used when a test protocol is generated for ATPG.

For ATPG, **test_bsd_default_strobe** defines, for the design under test, the default time at which values are strobed on the primary output ports and bidirectional ports in output mode. The value of this variable must be less than or equal to the clock period value.

The value of this variable affects the outcome of the **check_bsd** command and the test program produced by the **create_bsd_patterns** or **write_test** command. The **check_bsd** command uses the value of this variable to check a design against the design rules of a bsd methodology.

Changing the value of **test_bsd_default_strobe** modifies the content of the inferred or created test protocol generated by **check_bsd**. To generate the test program, **create_bsd_patterns** or **write_test** uses the value of **test_bsd_default_strobe** in the test protocol attached to the vector database (.vdb) file.

To determine the current value of this variable, type **printvar test_bsd_default_strobe**. For a list of all **test** variables and their current values, type **print_variable_group test**.

SEE ALSO

check_bsd(2)
write_test(2)
test_bsd_default_bidir_delay(3)
test_bsd_default_delay(3)
test_bsd_default_strobe_width(3)

test_bsd_default_strobe_width

Defines the default strobe pulse width, which is the default time that specifies how long after invocation the strobe pulse needs to be held active for bsd applications.

TYPE

positive nonzero integer

DEFAULT

0.0

GROUP

test_variables

DESCRIPTION

Defines the default strobe pulse width, which is the default time that specifies how long after invocation the strobe pulse needs to be held active for bsd applications.

Changing the value of this variable modifies the content of the inferred or created test protocol generated by the **check_bsd** command. The **create_bsd_patterns** or **write_test** command uses the value of this variable (as stated in the test protocol attached to the design) to generate the test program for vector formats that support the notion of strobe width. For those formats that do not support it, the strobe width value specified is ignored.

The value of the **test_bsd_default_strobe_width** variable is a real number value whose units are assumed to be nanoseconds, and the value must be a positive number. The sum of this value and the strobe time value must be less than or equal to the clock period value.

To determine the current value of this variable, use **printvar test_bsd_default_strobe_width**. For a list of all **test** variables and their current values, use the **print_variable_group test** command.

SEE ALSO

check_bsd(2)
write_test(2)
test_bsd_default_bidir_delay(3)
test_bsd_default_delay(3)
test_bsd_default_strobe(3)

test_bsd_manufacturer_id

Specifies the manufacturer ID to use to create the value captured in the device identification register during execution of the **insert_bsd** command.

TYPE

integer

DEFAULT

0

GROUP

bsd_variables

DESCRIPTION

Specifies the manufacturer ID to use to create the value captured in the device identification register during execution of the **insert_bsd** command. This unique ID represents the manufacturer ID assigned to the organization.

To determine the current value of this variable, type **printvar test_bsd_manufacturer_id**. For a list of **bsd** variables and their current values, type **print_variable_group bsd**.

SEE ALSO

test_bsd_part_number(3)
test_bsd_version_number(3)

test_bsd_optimize_control_cell

When true, allows the **optimize_bsd** command to optimize allocation of BSR control cells during area-driven optimization, using the value of the **test_bsd_control_cell_drive_limit** variable.

TYPE

Boolean

DEFAULT

false

GROUP

bsd_variables

DESCRIPTION

When true, allows the **optimize_bsd** command to optimize allocation of boundary-scan register (BSR) control cells during area-driven optimization, using the value of the **test_bsd_control_cell_drive_limit** variable. If the area constraints are not violated, this optimization does not take place. To specify area constraints for the current design, use the **set_max_area** command.

When *false* (the default), no optimization of BSR control cells takes place during area-driven optimization.

To determine the current value of this variable, type **printvar test_bsd_optimize_control_cell**. For a list of **bsd** variables and their current values, type **print_variable_group bsd**.

SEE ALSO

set_max_area(2)
test_bsd_allow_tolerable_violations(3)
test_bsd_control_cell_drive_limit(3)

test_bsd_part_number

Specifies the part number to use to create the value captured in the device identification register during execution of the **insert_bsd** command.

TYPE

integer

DEFAULT

0

GROUP

bsd_variables

DESCRIPTION

Specifies the part number to use to create the value captured in the device identification register during execution of the **insert_bsd** command. This number, which represents the part number assigned to the IC, should be unique inside the organization.

To determine the current value of this variable, type **printvar test_bsd_part_number**. For a list of **bsd** variables and their current values, type **print_variable_group bsd**.

SEE ALSO

test_bsd_version_number(3)
test_bsd_manufacturer_id(3)

test_bsd_version_number

Specifies the version number to use to create the value captured in the device identification register during execution of the **insert_bsd** command.

TYPE

integer

DEFAULT

0

GROUP

bsd_variables

DESCRIPTION

Specifies the version number to use to create the value captured in the device identification register during execution of the **insert_bsd** command. This represents the version number assigned to this IC and is unique for this IC.

To determine the current value of this variable, type **printvar test_bsd_version_number**. For a list of **bsd** variables and their current values, type **print_variable_group bsd**.

SEE ALSO

test_bsd_manufacturer_id(3)
test_bsd_part_number(3)

test_bsdl_default_suffix_name

Specifies the default suffix for the name of the BSDL file generated by the **write_bsdl** command.

TYPE

string

DEFAULT

bsdl

GROUP

bsd_variables

DESCRIPTION

Specifies the default suffix for the name of the Boundary-Scan Description Language (BSDL) file the **write_bsdl** command generates. If the **write_bsdl** command does not specify an output file, by default the output file is named *top_level_design_name.suffix*, *suffix* being the value of **test_bsdl_default_suffix_name**.

To determine the current value of this variable, type **printvar test_bsdl_default_suffix_name**. For a list of **bsd** variables and their current values, type **print_variable_group bsd**.

SEE ALSO

write_bsdl(2)

test_bsdl_max_line_length

Specifies the maximum number of characters per line for the output BSDL file the **write_bsdl** command produces.

TYPE

integer

DEFAULT

80

GROUP

bsd_variables

DESCRIPTION

Specifies the maximum number of characters per line for the output Boundary-Scan Description Language (BSDL) file the **write_bsdl** command produces. The maximum line length should not be less than 50 or more than 132. If you specify a line length outside this range, **write_bsdl** uses the default instead.

To determine the current value of this variable, type **printvar test_bsdl_max_line_length**. For a list of **bsd** variables and their current values, type **print_variable_group bsd**.

SEE ALSO

write_bsdl(2)

test_capture_clock_skew

Specifies a qualitative measure of clock skew.

TYPE

string

DEFAULT

small_skew

GROUP

test_variables

DESCRIPTION

Specifies a qualitative measure of clock skew. Values are *no_skew*, *small_skew* (the default), or *large_skew*.

Because the **check_test** command does not take into account the delays through capture paths or the skew between clocks, DFT Compiler might inadvertently merge capture groups that should remain separate, which results in badly generated patterns. The **test_capture_clock_skew** variable provides a coarse control over the inference process by letting you specify the amount of clock skew to be assumed.

no_skew

Directs DFT Compiler to place clocks with identical waveforms in the same capture group. This value assumes that waveforms on clock pins of cells are identical to the waveforms specified on the top-level ports. As a result, the sequencing of events on clock pins is the same as the sequencing of events on the top-level ports.
The value *no_skew* results in the fewest clock groups and, therefore, the smallest test program. However, the risk is greatest with *no_skew* because DFT Compiler might generate invalid expected responses. Use this value with caution; validate your vectors through simulation.

small_skew (the default):

Directs DFT Compiler to place two clocks in different capture groups if there is a path between the cells that launch and capture at the same time in the two clock domains. The *small_skew* value assumes that the waveforms on clock pins of cells are identical to the waveforms specified on the top-level ports, except for simultaneous events, which are slightly skewed. The skew is assumed to be smaller than any nonzero time differences specified on the ports of the design. *small_skew* differs from *no_skew* only for events that are simultaneous at the clock ports. Although the possibility of invalid vectors is not eliminated, it is greatly reduced.

large_skew

Directs DFT Compiler to place two clocks in different capture groups, regardless of waveforms, if there is a path between the two clock domains. This value assumes that the clock skew is arbitrarily large. The *large_skew* value results in the largest number of clock groups and, therefore, the largest test program. However, it completely eliminates the possibility of invalid vectors caused by skew. *large_skew* should be overly conservative for most purposes.

Note: This variable annotates the skew between independently controllable clocks. The clock skew between two capture cells in the same clock domain is assumed to be 0.

To determine the current value of this variable, type **printvar test_capture_clock_skew**. For a list of all **test** variables and their current values, type **print_variable_group test**.

SEE ALSO

test_cc_ir_masked_bits

Identifies instruction register (IR) bits to be masked during the search by the **check_bsd** command for all possible implemented instructions.

TYPE

integer

DEFAULT

0

GROUP

bsd_variables

DESCRIPTION

Identifies instruction register (IR) bits to be masked during the search by the **check_bsd** command for all possible implemented instructions. By default, no bits are masked.

Masking IR bits shortens the sequential search. The sequential search becomes exponentially costly as the IR length increases beyond 8 bits. The compliance checker masks bits that contain a 1 in the binary equivalent of the decimal integer in this variable. For example, if you want to mask bits 0 and 3, set the variable with decimal 9, the equivalent of binary 00...1001. Similarly, a value of 7 (binary 00...0111) masks bits 0, 1 and 2; 8 masks bit 3; and so on.

To determine the current value of this variable, type **printvar test_cc_ir_masked_bits**. For a list of bsd variables and their current values, type **print_variable_group bsd**.

SEE ALSO

check_bsd(2)
test_cc_ir_value_of_masked_bits(3)

test_cc_ir_value_of_masked_bits

Specifies values to be forced into bits of the instruction register (IR) that are masked, during the search by the **check_bsd** command for all possible implemented instructions.

TYPE

integer

DEFAULT

0

GROUP

bsd_variables

DESCRIPTION

Specifies values to be forced into bits of the instruction register (IR) that are masked, during the search by the **check-bsd** command for all possible implemented instructions. The value of the **test_cc_ir_masked_bits** variable identifies the bits to be masked. By default, *0*s are forced into all masked bits.

The compliance checker forces a 0 or a 1 into each masked bit, based on the binary equivalent of the decimal integer value of **test_cc_ir_value_of_masked_bits**, adjusted for the bit-width of the IR. For example, for a 4-bit IR, if bits 0 and 3 are masked and the value of **test_cc_ir_value_of_masked_bits** is 5 (binary 0101), bit 0 receives a value of 1 and bit 3 a value of 0.

To determine the current value of this variable, type **printvar test_cc_ir_value_of_masked_bits**. For a list of bsd variables and their current values, type **print_variable_group bsd**.

SEE ALSO

check_bsd(2)
test_cc_ir_masked_bits(3)

test_check_port_changes_in_capture

Checks (through the **check_test** command) for changes in values applied to bidirectional ports in the parallel measure cycle.

TYPE

Boolean

DEFAULT

true

GROUP

test_variables

DESCRIPTION

Checks (through the **check_test** command) for changes in values applied to bidirectional ports in the parallel measure cycle. Such changes can cause an unreliable capture in sequential cells. When the value is *false*, the checks are disabled.

The DFT Compiler automatic test pattern generation (ATPG), invoked by the **create_test_patterns** command, assumes that values applied to primary inputs in the parallel measure cycle are stable while response data is being measured at primary output ports and captured in scan (or valid nonscan) sequential cells. This assumption is also made for a bidirectional port when that port is used as an input. Therefore, if the value applied to a port in the parallel measure cycle changes before response data is captured into sequential cells, this invalidates the patterns generated by the **create_test_patterns** command.

The default test protocol inferred by the **check_test** command ensures that the values applied to primary inputs (and bidirectionals when used as inputs) in the parallel measure cycle do not change until response data is measured at primary outputs and captured into sequential cells. However, a change to the default test timing can cause **check_test** to infer a protocol, which changes the value applied to bidirectional ports before response data is captured into sequential cells.

For example, consider this case: The bidirectional delay is set to a value smaller than the active (capture) edge of the clock, the protocol value of a bidirectional port is *M* (output don’t care) in capture cycle, and the port can be used as an input by ATPG. This condition can cause a mismatch between the actual values captured into sequential cells driven by this port and the expected values of these sequential cells in the generated patterns. An input value is applied to the bidirectional port at the bidirectional delay time in the parallel measure cycle. However, the bidirectional port is released (changes to ’M’) in the capture cycle at the bidirectional delay time. This is before the capture edge of the clock.

While simulating the test protocol, **check_test** detects any change to the value applied to a bidirectional port in the parallel measure cycle that can cause an unreliable capture in the sequential cells driven by that port. All sequential cell pins affected by the change are marked as violated. The **create_test_patterns** command does not use the violated pins for observing fault effects and considers them as unknown when expecting response values.

Set this variable to *false* only after you have carefully analyzed the related unreliable capture warnings issued by **check_test** and have proven them too conservative. In general, setting this variable to *false* can lead to patterns that mismatch in simulation or on the tester.

For example, this variable can be set to *false* when the following conditions are present: The generated patterns are to be formatted in the LSI format, the design includes a three-state enable inhibit signal (input port) associated with the bidirectional ports, and this input port is correctly attributed with the "test_bidir_control[_inverted]" signal_type. The formatted vectors change the value of the enable inhibit signal between cycle 2 (parallel measure) and cycles 3 and 4 (capture), so that bidirectionals are forced into input mode in cycles 3 and 4. All output values driven by the bidirectional ports in cycle 2 are driven back in in cycles 3 and 4 when the bidirectional is forced into input mode. The default bidirectional delay can be set to a value less than the active edge of all clocks without any mismatches in the generated patterns.

To determine the current value of this variable, type prompt> **printvar test_check_port_changes_in_capture**

For a list of all **test** variables and their current values, type

prompt> **print_variable_group test**

SEE ALSO

test_clock_port_naming_style

Specifies the naming style used by the **insert_scan** command for global test signal ports created in designs during the addition of test circuitry.

TYPE

string

DEFAULT

test_c%s

GROUP

insert_dft_variables

DESCRIPTION

Specifies the naming style used by the **insert_scan** command for global test signal ports created in designs during the addition of test circuitry. If the **set_scan_signal** command or the **set_signal_type** command identifies suitable ports, new ports are not added.

This variable must contain one %s (percent s) character sequence. When adding a port, if the new port name is the same as an existing port name, that creates a unique name.

For example, if this variable is set to **MY_TC%s** when **insert_scan** adds a new port for the "test clock" signal, the port is named: MY_TC

If a port, **MY_TC** already exists in the design, the new port is named: MY_TCa

To determine the current value of this variable, use **printvar test_clock_port_naming_style**. For a list of all **insert_dft** variables and their current values, use the **print_variable_group insert_dft** command.

SEE ALSO

insert_dft_variables(3)

test_dedicated_subdesign_scan_outs

Instructs DFT Compiler to create dedicated scan-out ports on subdesigns.

TYPE

Boolean

DEFAULT

false

GROUP

insert_dft_variables

DESCRIPTION

Instructs DFT Compiler to create dedicated scan-out ports on subdesigns. The default is *false*. When *false*, DFT Compiler uses existing subdesign ports where possible. If *true* it creates new dedicated sub-design ports for scan.

This variable only affects cases where the Q output of flow is directly connected to output of a sub-block. If the variable is set to true, it will create a new output port driven by fanout from Q. This becomes a multiport net and buffers are added to fix this multiport net.

If the variable is set to false, in the above conditions the Q output gets resused as a scan-out.

If there is no flow with Q directly driving a sub-block output prot, DFTC will not reuse a functional prot and add a mux. Instead, it will create a new scan-out port.

To determine the current value of this variable, type **printvar test_dedicated_subdesign_scan_outs**. For a list of **insert_dft** variables and their current values, type **print_variable_group insert_dft**.

SEE ALSO

insert_dft(2)
insert_dft_variables(3)

test_default_bidir_delay

Defines the default switching time of bidirectional ports in a tester cycle.

TYPE

positive integer

DEFAULT

0.0

GROUP

test_variables

DESCRIPTION

Defines the default switching time of bidirectional ports in a tester cycle. The value is a positive real number in nanoseconds. This value is used when a test protocol is generated for ATPG.

For ATPG, **test_default_bidir_delay** defines for the design under test the following:

> The default time at which values are applied (driven) to the bidirectional ports in input mode during parallel measure cycle.
>
> The time at which bidirectional ports are released (undriven) during capture cycle.

The value of **test_default_bidir_delay** must be less than the output strobe time and less than the capture clock active edge value.

The value of this variable affects the outcome of the **dft_drc** command and the test protocol produced by the **write_test_protocol** command. The **dft_drc** command uses the value of this variable to check a design against the design rules of a scan test methodology.

Changing the value of **test_default_bidir_delay** modifies the content of the inferred or created test protocol generated by **dft_drc**. You can override the value of this variable by reading in a user-defined test protocol using the **read_test_protocol** command.

To determine the current value of this variable, type **printvar test_default_bidir_delay**. For a list of all **test** variables and their current values, type **print_variable_group test**.

SEE ALSO

dft_drc(2)

```
read_test_protocol(2)
write_test(2)
write_test_protocol(2)
port_attributes(3)
test_default_delay(3)
test_default_period(3)
test_default_strobe(3)
test_default_strobe_width(3)
```

test_default_delay

Defines the default time in a tester cycle to apply values to input ports.

TYPE

positive integer

DEFAULT

0.0

GROUP

test_variables

DESCRIPTION

Defines the default time in a tester cycle to apply values to input ports. The value is a positive real number in nanoseconds. The default value is 5. This variable is used to generate a protocol file for ATPG.

For ATPG, **test_default_delay** defines for the design under test the default time at which values are applied (driven) to the primary inputs. The value of **test_default_delay** must be less than the output strobe time, and less than the capture clock edge value.

The value of this variable affects the outcome of the **dft_drc** command and the test program produced by the **write_test** command. The **dft_drc** command uses the value of this variable to check a design against the design rules of a scan test methodology. Changing the value of **test_default_delay** modifies the content of the inferred or created test protocol generated by **dft_drc** or **create_test_protocol**. You can override the value of this variable by reading in a user-defined test protocol using **read_test_protocol**. To generate the test program, **write_test** uses the value of **test_default_delay** in the test protocol attached to the vector data base (.vdb) file.

To determine the current value of this variable, type **printvar test_default_delay**. For a list of all **test** variables and their current values, type **print_variable_group test**.

SEE ALSO

dft_drc(2)
read_test_protocol(2)
write_test(2)
write_test_protocol(2)
port_attributes(3)
test_default_bidir_delay(3)
test_default_period(3)
test_default_strobe(3)

```
test_default_strobe_width(3)
```

test_default_period

Defines the default length of a test vector cycle.

TYPE

positive nonzero integer

DEFAULT

100.0

GROUP

test_variables

DESCRIPTION

Defines the default length of a test vector cycle. This value translates directly to the speed of application of the test vectors on automated test equipment (ATE). For example, ATE running at 10 MHz would imply a period of 100 nanoseconds. Changing the value of this variable modifies the content of the inferred test protocol generated by the **dft_drc** command. You can overwrite the value of this variable by providing a user-defined test protocol and reading it in using the **read_test_protocol** command or the **create_test_clock** command. To generate the test program, the **write_test** command uses the value of this variable in the test protocol attached to the current design.

The value of the **test_default_period** variable is a real number whose units are assumed to be nanoseconds, and the value must be a positive number.

To determine the current value of this variable, use **printvar test_default_period**. For a list of all **test** variables and their current values, use the **print_variable_group test** command.

SEE ALSO

dft_drc(2)
read_test_protocol(2)
write_test(2)
write_test_protocol(2)
test_default_bidir_delay(3)
test_default_delay(3)
test_default_strobe(3)
test_default_strobe_width(3)

test_default_scan_style

Defines the default scan style for the **insert_dft** command if a scan style is not specified with the **set_scan_style** command.

TYPE

string

DEFAULT

multiplexed_flip_flop

GROUP

test_variables

DESCRIPTION

Defines the default scan style for the **insert_dft** command if a scan style is not specified with the **set_scan_style** command.

This variable must identify one of the following supported scan styles: *multiplexed_flip_flop*, *clocked_scan*, *lssd*, *aux_clock_lssd*, *combinational*, or *none*. The default is *multiplexed_flip_flop*.

To determine the current value of this variable, type **printvar test_default_scan_style**. For a list of **test** variables and their current values, type **print_variable_group test**.

SEE ALSO

insert_dft(2)
test_variables(3)

test_default_strobe

Defines the default strobe time in a test cycle for output ports and bidirectional ports in output mode.

TYPE

positive nonzero integer

DEFAULT

40.0

GROUP

test_variables

DESCRIPTION

Defines the default strobe time in a test cycle for output ports and bidirectional ports in output mode. The value is a positive real number in nanoseconds. This value is used when a test protocol is generated for ATPG.

For ATPG, **test_default_strobe** defines, for the design under test, the default time at which values are strobed on the primary output ports and bidirectional ports in output mode. The value of this variable must be less than or equal to the clock period value.

The value of this variable affects the outcome of the **dft_drc** command and the test program produced by the **write_test** command. The **dft_drc** command uses the value of this variable to check a design against the design rules of a scan test methodology.

Changing the value of **test_default_strobe** modifies the content of the inferred or created test protocol generated by **dft_drc**. You can override the value of this variable for by reading in a user-defined test protocol using the **read_test_protocol** command. To generate the test program, **write_test** uses the value of **test_default_strobe** in the test protocol attached to the vector database (.vdb) file.

To determine the current value of this variable, type **printvar test_default_strobe**. For a list of all **test** variables and their current values, type **print_variable_group test**.

SEE ALSO

dft_drc(2)
read_test_protocol(2)
write_test(2)
write_test_protocol(2)
test_default_bidir_delay(3)
test_default_delay(3)

```
test_default_period(3)
test_default_strobe_width(3)
```

test_default_strobe_width

Defines the default strobe pulse width, which is the default time that specifies how long after invocation the strobe pulse needs to be held active.

TYPE

positive integer

DEFAULT

0.0

GROUP

test_variables

DESCRIPTION

Defines the default strobe pulse width, which is the default time that specifies how long after invocation the strobe pulse needs to be held active.

Changing the value of this variable modifies the content of the inferred or created test protocol generated by the **dft_drc** command. You can overwrite the value of this variable by providing a user-defined test protocol and reading it in using the **read_test_protocol** command. The **write_test** command uses the value of this variable (as stated in the test protocol attached to the design) to generate the test program for vector formats that support the notion of strobe width. For those formats that do not support it, the strobe width value specified is ignored.

The value of the **test_default_strobe_width** variable is a real number value whose units are assumed to be nanoseconds, and the value must be a positive number. The sum of this value and the strobe time value must be less than or equal to the clock period value.

To determine the current value of this variable, use **printvar test_default_strobe_width**. For a list of all **test** variables and their current values, use the **print_variable_group test** command.

SEE ALSO

dft_drc(2)
write_test(2)
read_test_protocol(2)
write_test_protocol(2)
test_default_bidir_delay(3)
test_default_delay(3)
test_default_period(3)
test_default_strobe(3)

test_design_analyzer_uses_insert_scan

Executes (when *true*) the **insert_scan** command through a Design Analyzer menu.

TYPE

string

DEFAULT

true

GROUP

view_variables

DESCRIPTION

Executes (when *true*) the **insert_scan** command through a Design Analyzer menu. When *true* (the default), Design Analyzer executes the **insert_scan** command when you perform the following:

```
     From the Design Analyzer text window,
     select Tools > Test Synthesis > Insert Internal Scan Circuitry.

     Click OK.
```

When this variable's value is *false*, Design Analyzer executes the **insert_dft** command, instead of **insert_scan**.

You cannot pass maximum scan chain length options to the **insert_scan** command. The **EQN-18** error message appears if you try.

To determine the current value of this variable, type **printvar test_design_analyzer_uses_insert_scan**. For a list of **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

insert_scan (2), **insert_dft** (2); **view_variables** (3).

test_disable_find_best_scan_out

Selects the scan-out pin on a scan cell based on availability instead of timing slack.

TYPE

Boolean

DEFAULT

false

GROUP

insert_dft_variables

DESCRIPTION

When set to false (the default), the **insert_scan** command and the **insert_dft** command select the scan-out pin on a scan cell that has the greatest timing slack. This usually means that **insert_scan** and **insert_dft** make extensive use of Qbar pins to drive the scan-in pins of scan cells. DFT Compiler supports inversions on the scan chain.

If you are using tools that do not support inversions, you can disable this behavior by setting this variable to true. The **insert_dft** command does not attempt to find the optimum driver. Instead, the tool selects the driver based on availability, and it chooses the dedicated scan-out driver if any are defined in the library. The chosen driver can still be inverting or noninverting.

Use the following command to determine the current value of this variable:

```
   prompt> printvar test_disable_find_best_scan_out
```

For a list of **insert_dft** variables and their current values, enter the following command:

```
   prompt> print_variable_group insert_dft
```

SEE ALSO

insert_dft(2)
insert_dft_variables(3)

test_dont_fix_constraint_violations

Minimizes performance constraint violations.

TYPE

Boolean

DEFAULT

false

GROUP

insert_dft_variables

DESCRIPTION

Minimizes performance constraint violations. When the value of this variable is set to *false* (the default), the **insert_scan** command and the **insert_dft** command attempt to minimize performance constraint violations. You can disable this behavior by setting the variable *true*.

To determine the current value of this variable, type **printvar test_dont_fix_constraint_violations**. For a list of **insert_dft** variables and their current values, type **print_variable_group insert_dft**.

SEE ALSO

insert_dft(2)
insert_dft_variables(3)

test_enable_capture_checks

Controls checking for capture violations during execution of the **check_dft** command.

TYPE

Boolean

DEFAULT

true

GROUP

test_variables

DESCRIPTION

Use this variable to control the checking for capture violations, during the execution of the **check_dft** command.

Setting the value of **test_enable_capture_checks** to *true* (the default value) enables capture checks during **check_dft** command activity.

Setting the value of **test_enable_capture_checks** to *false* disables capture checks during **check_dft** command activity.

To determine the current value of this variable, type **printvar test_enable_capture_checks**. For a list of **test_variables** variables and their current values, type **print_variable_group test**.

SEE ALSO

test_variables(3)

test_infer_slave_clock_pulse_after_capture

Guides protocol inference for master/slave test design methodologies during execution of the **check_test** command.

TYPE

string

DEFAULT

infer

GROUP

test_variables

DESCRIPTION

Guides protocol inference for master/slave test design methodologies during execution of the **check_test** command.

When set to *infer* (the default value), **check_test** protocol inference is based on an analysis of the scan cell states instead of the scan style (as previously done in the 1997.01 and earlier releases). The other possible values are *pulse* and *no_pulse*. If you set the variable to *pulse*, all slave clocks are pulsed after capture. If you set it to *no_pulse*, no slave clocks are pulsed after capture. If a slave clock is being inferred, and the value of the variable is invalid, the warning message TEST-314 is issued and the compiler uses the *infer* value by default.

To determine the current value of this variable, type **printvar test_infer_slave_clock_pulse_after_capture**. For a list of **test_variables** variables and their current values, type **print_variable_group test**.

SEE ALSO

test_variables(3)

test_isolate_hier_scan_out

Prevents the **insert_dft** command inserting logic that isolates scan connections at hierarchical boundaries during functional operation.

TYPE

integer

DEFAULT

0

GROUP

insert_dft_variables

DESCRIPTION

Prevents the **insert_dft** command inserting logic that isolates scan connections at hierarchical boundaries, during functional operation. When value is set to *1*, the **insert_dft** command inserts logic that isolates scan connections at hierarchical boundaries during functional operation. This can reduce dynamic switching currents and output loading. When set to *0* (the default), no logic is inserted.

Note that this variable does not affect **insert_scan** commands.

To determine the current value of this variable, type **printvar test_isolate_hier_scan_out**. For a list of **insert_dft** variables and their current values, type **print_variable_group insert_dft**.

SEE ALSO

insert_dft(2)
insert_dft_variables(3)

test_jump_over_bufs_invs

Determines whether or not **insert_scan** and **preview_scan** consider output pins of buffers and inverters to be internal clocks.

TYPE

Boolean

DEFAULT

true

GROUP

test_variables

DESCRIPTION

Determines whether or not **insert_scan** and **preview_scan** consider output pins of buffers and inverters to be internal clocks. When true (the default), output pins of only multiple-input gates, and not of buffers and inverters, are considered to be internal clocks. When false, output pins of buffers and inverters are considered to be internal clocks.

The scan architect called in both commands looks in the design to treat internal clocks as separate clocks.

If the **-internal_clocks** option of the **set_scan_configuration** command is set to true, the output pins of multiple-input gates are considered separate clocks. Setting **test_jump_over_bufs_invs** to false increases the granularity of the search by allowing buffers and inverters output pins to be considered as separate clocks. For example, setting the variable to false allows you to treat each branch of a clock tree as a different clock domain.

To determine the current value of this variable, type **printvar test_jump_over_bufs_invs**. For a list of all **test** variables and their current values, type **print_variable_group test**.

SEE ALSO

set_scan_configuration(2)

test_mode_port_inverted_naming_style

Specifies the naming style to use for the **test_hold_logic_zero** type of test mode signal ports to be created in the design.

TYPE

string

DEFAULT

test_mode_i%s

GROUP

insert_dft_variables

DESCRIPTION

Specifies the naming style to use for the **test_hold_logic_zero** type of test mode signal ports to be created in the design. (New ports are not created if suitable ports are identified with the **set_dft_signal** command.)

This variable must contain one %s (percent s) character sequence. When a new port is added, if its name is the same as that of an existing port, that creates a unique name.

For example, if this variable is set to MY_TM_I%s, when synthesis adds a new port for the test mode signal, the port is named

MY_TM_I

If a port named MY_TM_I already exists in the design, the new port is named

MY_TM_Ia

To determine the current value of this variable, use **printvar test_mode_port_inverted_naming_style**. For a list of all **insert_dft** variables and their current values, use the **print_variable_group insert_dft** command.

SEE ALSO

insert_dft(2)
insert_dft_variables(3)

test_mode_port_naming_style

Determines the naming style to be used by **insert_dft** command for test mode ports created in designs during the addition of test point circuitry.

TYPE

string

DEFAULT

test_mode%s

GROUP

insert_dft_variables

DESCRIPTION

Determines the naming style to be used by **insert_dft** command for test mode ports created in designs during the addition of test point circuitry. New ports are not created if suitable ports are identified with the **set_dft_signal** command.

This variable must contain one %s (percent s) character sequence. When a port is added, if its name is the same as that of an existing port, that creates a unique name.

For example, if this variable is set to MY_TM%s, when synthesis adds a new port for the test mode signal, the port is named

MY_TM

If a port, MY_TM already exists in the design, the new port is named

MY_TMa

To determine the current value of this variable, use **printvar test_mode_port_naming_style**. For a list of all **insert_dft** variables and their current values, use the **print_variable_group insert_dft** command.

SEE ALSO

insert_dft(2)
insert_dft_variables(3)

test_mux_constant_si

Specifies how scan insertion uses a port you declare as scan input, when the port is tied high or to the ground in functional mode.

TYPE

Boolean

DEFAULT

false

GROUP

insert_dft_variables

DESCRIPTION

Specifies how scan insertion uses a port you declare as scan input, when the port is tied high or to the ground in functional mode. When you set this variable to *false* (the default value), scan insertion ignores the tie-off logic and directly uses the port as a scan input. This might change the output of the design during functional mode. When you set this variable to *true*, scan insertion multiplexes the scan input signal with the constant logic, using the scan enable signal to control the multiplexer.

To determine the current value of this variable, type **printvar test_mux_constant_si**. For a list of **insert_dft** variables and their current values, type **print_variable_group insert_dft**.

SEE ALSO

insert_dft(2)
insert_dft_variables(3)

test_mux_constant_so

Specifies how scan insertion uses a port you declare as scan output, when the port is tied high or to the ground in functional mode.

TYPE

Boolean

DEFAULT

false

GROUP

insert_dft_variables

DESCRIPTION

Specifies how scan insertion uses a port you declare as scan output, when the port is tied high or to the ground in functional mode. When you set this variable to *false* (the default value), scan insertion ignores the tie-off logic and directly uses the port as a scan output. This might change the output of the design during functional mode. When you set this variable to *true*, scan insertion multiplexes the scan output signal with the constant logic, using the scan enable signal to control the multiplexer.

To determine the current value of this variable, type **printvar test_mux_constant_so**. For a list of **insert_dft** variables and their current values, type **print_variable_group insert_dft**.

SEE ALSO

insert_dft(2)
insert_dft_variables(3)

test_non_scan_clock_port_naming_style

Specifies the style the **insert_dft** command uses to name the ports that clock gating creates for nonscan clocks.

TYPE

string

DEFAULT

test_nsc_%s

GROUP

insert_dft_variables

DESCRIPTION

Specifies the style the **insert_dft** command uses to name the ports that clock gating creates for nonscan clocks.

This variable must contain one %s (percent s) character sequence. The %s is replaced by a string identifying the original clock, its inversion (either *i* for inverted or *n* for noninverted), and the inactive level (either 0 or 1).

The following example shows the renaming of this variable from its setting of MY_NSC_%s to MY_NSC_clockn0: When **insert_dft** creates a nonscan clock (from an original clock named "clock") that is noninverted and inactive low, that port is named

```
MY_NSC_clockn0
```

To determine the current value of this variable, type **printvar test_clock_port_naming_style**. For a list of all **insert_dft** variables and their current values, type **print_variable_group insert_dft**.

SEE ALSO

insert_dft(2)

test_point_keep_hierarchy

Synthesizes (when *true*) test points and ungroups the test point design, during execution of the **insert_dft** command.

TYPE

string

DEFAULT

false

GROUP

insert_dft_variables

DESCRIPTION

Synthesizes (when *true*) test points and ungroups the test point design, during execution of the **insert_dft** command, when set to *false*, the default. When set to *true*, **insert_dft** keeps each test point design in a separate level of hierarchy. The *true* setting also prevents **insert_dft** from optimizing the test point logic.

To determine the current value of this variable, type **printvar test_point_keep_hierarchy**. For a list of all **insert_dft** variables and their current values, type **print_variable_group insert_dft**.

SEE ALSO

preview_dft(2)
insert_dft(2)

test_preview_scan_shows_cell_types

Shows (when *true*) cell instance types, during execution of the **preview_scan** command.

TYPE

string

DEFAULT

false

GROUP

preview_scan_variables

DESCRIPTION

Shows (when *true*) cell instance types, during execution of the **preview_scan** command. When set to *false* (the default), cell type information is suppressed.

To determine the current value of this variable, type **printvar test_preview_scan_shows_cell_types**. For a list of **preview_scan** variables and their current values, type **print_variable_group preview_scan**.

SEE ALSO

preview_scan_variables(3)

test_protocol_add_cycle

Adds an extra cycle (when *true*) after the shift cycle, in the test protocol, during execution of the **check_test** command.

TYPE

string

DEFAULT

true

GROUP

test_variables

DESCRIPTION

Adds an extra cycle (when *true*) after the shift cycle, in the test protocol, during execution of the **check_test** command. This occurs if you specify design bidirectional ports as input ports during the scan shift and the value is set to *true* (the default).

In the extra cycle, all bidirectional ports are set to output mode.

If you do not want **check_test** to add the extra cycle, set this variable to *false*.

The reason for adding this extra cycle is to prevent the **create_test_patterns** command from generating vectors that could cause contention on bidirectional ports.

To determine the current value of this variable, type **printvar test_protocol_add_cycle**. For a list of all **test** variables and their current values, type **print_variable_group test**.

SEE ALSO

write_test_protocol(2)

test_rtldrc_latch_check_style

Specifies the latch check style to use during **rtldrc** command activities.

TYPE

string

DEFAULT

default

GROUP

test_variables

DESCRIPTION

Specifies the latch check style to use during **rtldrc** command activities. The value of this variable affects the outcome of the **rtldrc** command, which uses this variable to determine the way latches are checked for violations against the design rules of a scan test methodology. When latch transparency checks are selected, nontransparent latches are flagged as violations (**TEST-1211**). This also suppresses checking for hold data violations (**TEST-965**) and latch controllability violations (**TEST-1210**).

To determine the current value of this variable, enter one of the following commands, depending on which mode you are using:

```
      prompt> printvar test_rtldrc_latch_check_style
      or
            dc_shell-t> printvar test_rtldrc_latch_check_style
```

SEE ALSO

set_scan_configuration(2)
test_default_scan_style(3)

test_scan_clock_a_port_naming_style

Determines the naming style to be used by the **insert_scan** command for test scan clock a ports created in designs during the addition of test scan circuitry.

TYPE

string

DEFAULT

test_sca%s

GROUP

insert_dft_variables

DESCRIPTION

Determines the naming style to be used by the **insert_scan** command for test scan clock a ports created in designs during the addition of test scan circuitry. (This variable is used in the same way as the **test_clock_port_naming_style** variable is used.) If the **set_scan_signal** command or the **set_signal_type** command identifies suitable ports, new ports are not added.

This variable must contain one %s (percent s) character sequence. When adding a port, if the new port name is the same as an existing port name, that creates a unique name.

For example, if this variable is set to **MY_TSCA%s** when **insert_scan** adds a new port for the "test scan clock a" signal, the port is named: MY_TSCA

If a port **MY_TSCA** already exists in the design, the new port is named: MY_TSCAa

To determine the current value of this variable, use **printvar test_scan_clock_a_port_naming_style**. For a list of all **insert_dft** variables and their current values, use the **print_variable_group insert_dft** command.

SEE ALSO

insert_dft(2)
test_clock_port_naming_style(3)
test_scan_clock_port_naming_style(3)
test_scan_clock_b_port_naming_style(3)
test_scan_enable_inverted_port_naming_style(3)
test_scan_enable_port_naming_style(3)

test_scan_clock_b_port_naming_style

Determines the naming style to be used by the **insert_scan** command for test scan clock b ports created in designs during the addition of test scan circuitry.

TYPE

string

DEFAULT

test_scb%s

GROUP

insert_dft_variables

DESCRIPTION

Determines the naming style to be used by the **insert_scan** command for test scan clock b ports created in designs during the addition of test scan circuitry. This variable must contain one %s (percent s) character sequence. When adding a port, if the new port name is the same as an existing port name, that creates a unique name.

For example, if this variable is set to **MY_TSCB%s** when **insert_scan** adds a new port for the "test scan clock b" signal, the port is named: MY_TSCB

If a port **MY_TSCB** already exists in the design, the new port is named: MY_TSCBa

To determine the current value of this variable, use **printvar test_scan_clock_b_port_naming_style**. For a list of all **insert_dft** variables and their current values, use the **print_variable_group insert_dft** command.

SEE ALSO

insert_dft(2)
test_clock_port_naming_style(3)
test_scan_clock_port_naming_style(3)
test_scan_clock_a_port_naming_style(3)
test_scan_enable_inverted_port_naming_style(3)
test_scan_enable_port_naming_style(3)

test_scan_clock_port_naming_style

Determines the naming style used by the **insert_scan** command for global test scan signal ports created in designs during the addition of test circuitry.

TYPE

string

DEFAULT

test_sc%s

GROUP

insert_dft_variables

DESCRIPTION

Determines the naming style used by the **insert_scan** command for global test scan signal ports created in designs during the addition of test circuitry. (This variable is used in the same way as the **test_clock_port_naming_style** variable is used.) This variable must contain one %s (percent s) character sequence. When adding a port, if the new port name is the same as an existing port name, that creates a unique name.

For example, if this variable is set to **MY_TSC%s** when **insert_scan** adds a new port for the "test scan clock" signal, the port is named: MY_TSC

If a port **MY_TSC** already exists in the design, the new port is named: MY_TSCa

To determine the current value of this variable, use **printvar test_scan_clock_port_naming_style**. For a list of all **insert_dft** variables and their current values, use the **print_variable_group insert_dft** command.

SEE ALSO

insert_dft(2)
test_clock_port_naming_style(3)
test_scan_clock_a_port_naming_style(3)
test_scan_clock_b_port_naming_style(3)
test_scan_enable_inverted_port_naming_style(3)
test_scan_enable_port_naming_style(3)

test_scan_enable_inverted_port_naming_style

Determines the naming style to be used by the **insert_scan** command for scan enable inverted ports created in designs during the addition of test scan circuitry.

TYPE

string

DEFAULT

test_sei%s

GROUP

insert_dft_variables

DESCRIPTION

Determines the naming style to be used by the **insert_scan** command for scan enable inverted ports created in designs during the addition of test scan circuitry. (This variable is used in the same way as the **test_clock_port_naming_style** variable is used.) This variable must contain one %s (percent s) character sequence. When adding a port, if the new port name is the same as an existing port name, that creates a unique name.

For example, if this variable is set to **MY_TSEI%s** when **insert_scan** adds a new port for the "test scan enable inverted" signal, the port is named: MY_TSEI

If a port **MY_TSEI** already exists in the design, the new port is named: MY_TSEIa

To determine the current value of this variable, use **printvar test_scan_enable_inverted_port_naming_style**. For a list of all **insert_dft** variables and their current values, use the **print_variable_group insert_dft** command.

SEE ALSO

insert_dft(2)
test_clock_port_naming_style(3)
test_scan_clock_port_naming_style(3)
test_scan_clock_a_port_naming_style(3)
test_scan_clock_b_port_naming_style(3)
test_scan_enable_port_naming_style(3)

test_scan_enable_port_naming_style

Determines the naming style to be used by the **insert_scan** command for test scan enable ports created in designs during the addition of test scan circuitry.

TYPE

string

DEFAULT

test_se%s

GROUP

insert_dft_variables

DESCRIPTION

Determines the naming style to be used by the **insert_scan** command for scan enable ports created in designs during the addition of test scan circuitry. (This variable is used in the same way as the **test_clock_port_naming_style** variable is used.) This variable must contain one %s (percent s) character sequence. When adding a port, if the new port name is the same as an existing port name, that creates a unique name.

For example, if this variable is set to **MY_TSE%s** when **insert_scan** adds a new port for the "test scan enable" signal, the port is named: MY_TSE

If a port **MY_TSE** already exists in the design, the new port is named: MY_TSEa

To determine the current value of this variable, use **printvar test_scan_enable_port_naming_style**. For a list of all **insert_dft** variables and their current values, use the **print_variable_group insert_dft** command.

SEE ALSO

insert_dft(2)
test_clock_port_naming_style(3)
test_scan_clock_port_naming_style(3)
test_scan_clock_a_port_naming_style(3)
test_scan_clock_b_port_naming_style(3)
test_scan_enable_inverted_port_naming_style(3)

test_scan_in_port_naming_style

Specifies the naming style used by the **insert_scan** command for serial test-signal ports created in designs during the addition of test circuitry.

TYPE

string

DEFAULT

test_si%s%s

GROUP

insert_dft_variables

DESCRIPTION

Specifies the naming style used by the **insert_scan** command for serial test-signal ports created in designs during the addition of test circuitry. (New ports are not added if suitable ports are identified with the **set_scan_signal** command or the **set_signal_type** command.)

This variable must contain two %s (percent s) character sequences. For a design with multiple scan chains, the index of the connecting scan chain replaces the first %s. For designs with single scan chains, %s is replaced by an empty string (""). The second %s is replaced by an alphabetic character if the new port name conflicts with an existing name.

For example, if this variable is set to SI_%s%s, when **insert_scan** adds a new port for the only scan-in signal of a design with one scan chain, the port is named

SI

If this variable is set to TDI_%s%s, when **insert_scan** adds a new port for the scan-in signal of the second scan chain in the design, the port is named

TDI_2

To determine the current value of this variable, use **printvar test_scan_in_port_naming_style**. For a list of **insert_dft** variables and their current values, use the **print_variable_group insert_dft** command.

SEE ALSO

insert_dft_variables(3)

test_scan_link_so_lockup_key

Indicates to the **preview_scan** command what key to use to identify cells with scan-out lock-up latches in reports.

TYPE

string

DEFAULT

l

GROUP

preview_scan_variables

DESCRIPTION

Indicates to the **preview_scan** command what key to use to identify cells with scan-out lock-up latches in reports. The default is *l*.

To determine the current value of this variable, type **printvar test_scan_link_so_lockup_key**. For a list of **preview_scan** variables and their current values, type **print_variable_group preview_scan**.

SEE ALSO

preview_scan_variables(3)

test_scan_link_wire_key

Indicates to the **preview_scan** command the key to use to identify cells that drive wire-scan links in reports.

TYPE

string

DEFAULT

w

GROUP

preview_scan_variables

DESCRIPTION

Indicates to the **preview_scan** command the key to use to identify cells that drive wire-scan links in reports. The default is *w*.

To determine the current value of this variable, type **printvar test_scan_link_wire_key**. For a list of **preview_scan** variables and their current values, type **print_variable_group preview_scan**.

SEE ALSO

preview_scan_variables(3)

test_scan_out_port_naming_style

Used the same as **test_scan_in_port_naming_style** .

TYPE

string

DEFAULT

test_so%s%s

GROUP

insert_dft_variables

DESCRIPTION

Specifies the naming style used by the **insert_scan** command for serial test-signal ports created in designs during the addition of test circuitry. (New ports are not added if suitable ports are identified with the **set_scan_signal** command or the **set_signal_type** command.)

This variable must contain two %s (percent s) character sequences. For a design with multiple scan chains, the index of the connecting scan chain replaces the first %s. For designs with single scan chains, %s is replaced by an empty string (""). The second %s is replaced by an alphabetic character if the new port name conflicts with an existing name.

For example, if this variable is set to SO_%s%s, when **insert_scan** adds a new port for the only scan-out signal of a design with one scan chain, the port is named

SO

If this variable is set to TDO_%s%s, when **insert_scan** adds a new port for the scan-out signal of the second scan chain in the design, the port is named

TDO_2

To determine the current value of this variable, use **printvar test_scan_out_port_naming_style**. For a list of **insert_dft** variables and their current values, use the **print_variable_group insert_dft** command.

SEE ALSO

insert_dft_variables(3)

test_scan_segment_key

Tells the **preview_scan** command what key to use to identify scan segments in reports.

TYPE

string

DEFAULT

s

GROUP

preview_scan_variables

DESCRIPTION

Tells the **preview_scan** command what key to use to identify scan segments in reports. The default is *s*.

To determine the current value of this variable, type **printvar test_scan_segment_key**. For a list of **preview_scan** variables and their current values, type **print_variable_group preview_scan**.

SEE ALSO

preview_scan_variables(3)

test_scan_true_key

Specifies to the **preview_scan** command the key to use to identify in reports cells with true scan attributes.

TYPE

string

DEFAULT

t

GROUP

preview_scan_variables

DESCRIPTION

Specifies to the **preview_scan** command the key to use to identify in reports cells with true scan attributes. The default value is *t*.

To determine the current value of this variable, type **printvar test_scan_true_key**. For a list of **preview_scan** variables and their current values, type **print_variable_group preview_scan**.

SEE ALSO

preview_scan_variables(3)

test_setup_additional_clock_pulse

When this variable is set to true, an extra clock cycle is added to all clocks in the design in the initialization procedure during execution of the **create_test_protocol** command.

TYPE

Boolean

DEFAULT

false

GROUP

test_variables

DESCRIPTION

When this variable is set to true, an extra clock cycle is added to all clocks in the design in the initialization procedure during execution of the **create_test_protocol** command.

You should set this variable to true if you are using clock gating and if the off-state of the clock driving the clock gating (at time=0) is the opposite of the active state of the clock gating latche enable input.

The extra cycle ensures that all clock gating latches are at a known state at the end of the initialization procedure.

To determine the current value of this variable, type **printvar test_setup_additional_clock_pulse**. For a list of all **test** variables and their current values, type **print_variable_group test**.

SEE ALSO

dft_drc(2)
create_test_protocol(2)
read_test_protocol(2)
write_test_protocol(2)

test_simulation_library

Indicates the pathname to a Verilog simulation library.

TYPE

list

GROUP

test_variables

DESCRIPTION

Use the **test_simulation_library** variable to specify the filenames of a Verilog simulation library, which is necessary if your design contains black-box elements like memories.

You could set the simulation library as follows and also specify your search path:

```
test_simulation_library = {foo.v bar.v}
search_path = { . synopsys_root + /libraries/syn}
synthetic_library = synthetic_library + {dft_lbist.sldb dw_foundation.sldb}
```

In above example, all the files named foo.v or bar.v under the specified search_path are considered as simulation library.

The **test_simulation_library** variable also accepts asterisk (*) as wildcard in the filenames to simplify the case when multiple files under a certain directory are simulation library files:

```
test_simulation_library = {/root/install/xyz/lib/verilog/*.v}
search_path = { . synopsys_root + /libraries/syn}
```

In above example, all the files with a ".v" name extension under /root/install/xyz/lib/verilog/ directory and are considered as simulation library. The asterisk (*) is the only wildcard character allowed in the library file name. If wildcard is used in the filename, it is better to specify the full pathname of the library files. Otherwise the combined effect of search_path and wildcard may bring in unexpected files as simulation library.

SEE ALSO

test_stil_max_line_length

Specifies the maximum line length for the file written by the **write_test_protocol -format stil** command.

TYPE

integer

DEFAULT

72

GROUP

test_variables

DESCRIPTION

Specifies the maximum line length for the file written by the **write_test_protocol -format stil** command. The recognized value is any integer between 60 and 5000. The default value is *72*.

When **write_test_protocol -format stil** writes the *design_name*.spf file, the command inserts a new line at or before the value of this variable, but identifiers and keywords are always preserved. If an identifier is longer than the value of this variable, the command makes an exception to the line length limit so as to preserve the identifier. The identifier begins at a new line and is printed in its entirety before the next new line.

To determine the current value of this variable, enter one of the following commands (depending on which mode you are using):

```
prompt> printvar test_stil_max_line_length
```

or

```
dc_shell-t> printvar test_stil_max_line_length
```

SEE ALSO

write_test_protocol(2)
test_variables(3)

test_stil_multiclock_capture_procedures

Indicates to the **write_test_protocol -format stil** command to create capture procedures in the STIL protocol, with multiple clocks active in each procedure.

TYPE

Boolean

DEFAULT

false

GROUP

test_variables

DESCRIPTION

Indicates to the **write_test_protocol -format stil** command to create capture procedures in the STIL protocol, with multiple clocks active in each procedure. The groups of clocks (capture clock groups) are determined by the **check_test** command. When this variable is *false* (the default), **write_test_protocol** places only one clock pulse in each capture clock procedure.

To determine the current value of this variable, type **printvar test_stil_multiclock_capture_procedures**. For a list of test variables and their current values, type **print_variable_group test**.

SEE ALSO

write_test_protocol(2)
test_variables(3)

test_stil_netlist_format

Indicates to the **write_test_protocol** command what netlist format to use when writing out STIL protocol files.

TYPE

string

DEFAULT

db

GROUP

test_variables

DESCRIPTION

Indicates to the **write_test_protocol** command what netlist format to use when writing out STIL protocol files. Allowed values are *db* (the default), *verilog*, or *vhdl*. Set this variable to match the netlist format of your design before issuing **write_test_protocol**, so that the port names in the STIL protocol file match the port names in the netlist.

In addition, you should also set either the **vhdlout_single_bit** variable or the **verilogout_single_bit** variable, as appropriate, before issuing the **write_test_protocol** command or the **write** command. These two variables control the handling of buses in both the STIL protocol and the netlist.

To determine the current value of this variable, type **printvar test_stil_netlist_format**. For a list of test variables and their current values, type **print_variable_group test**.

SEE ALSO

write(2)
write_test(2)
write_test_protocol(2)
verilogout_single_bit(3)
vhdlout_single_bit(3)

test_use_test_models

Indicates to DFT Compiler to create or use test models when performing scan insertion.

TYPE

string

DEFAULT

false

GROUP

test_variables

DESCRIPTION

Setting the **test_use_test_models** variable to *true* causes DFT Compiler to create test models or use test models, if they are present, when performing scan insertion. When the value is set as *false* (the default), DFT Compiler does not create test models when performing scan insertion.

When working with large designs, using the hierarchical scan synthesis DFT flow helps to reduce memory usage and speed up run time. Test models used by the hierarchical scan synthesis flow describe only the portions of subdesigns that are important for inserting DFT.

The **test_use_test_models** variable must be set for LBIST operation.

SEE ALSO

test_user_defined_instruction_naming_style

Indicates to the **check_bsd** command and the **write_bsdl** command the naming style to use for the user-defined (nonstandard) instructions inferred by these commands.

TYPE

string

DEFAULT

USER%d

GROUP

bsd_variables

DESCRIPTION

Indicates to the **check_bsd** command and the **write_bsdl** command the naming style to use for the user-defined (nonstandard) instructions inferred by these commands. The format for the specification is *string*%d; the specification must contain exactly one %d. Names resulting from the default specification (*USER%d*) are USER1, USER2, and so on.

To determine the current value of this variable, type **printvar test_user_defined_instruction_naming_style**. For a list of **bsd** variables and their current values, type **print_variable_group bsd**.

SEE ALSO

check_bsd(2)
write_bsdl(2)
bsd_variables(3)

test_user_test_data_register_naming_style

Indicates to the **check_bsd** command and the **write_bsdl** command the naming style to use for the user-defined (nonstandard) test data registers inferred by these commands.

TYPE

string

DEFAULT

UTDR%d

GROUP

bsd_variables

DESCRIPTION

Indicates to the **check_bsd** command and the **write_bsdl** command the naming style to use for the user-defined (nonstandard) test data registers inferred by these commands. The format for the specification is *string*%d; the specification must contain exactly one %d. Names resulting from the default specification (*UTDR%d*) are UTDR1, UTDR2, and so on.

To determine the current value of this variable, type **printvar test_user_test_data_register_naming_style**. For a list of **bsd** variables and their current values, type **print_variable_group bsd**.

SEE ALSO

check_bsd(2)
write_bsdl(2)
bsd_variables(3)

test_variables

SYNTAX

string **test_allow_clock_reconvergence** = "true"
string **test_capture_clock_skew** = "small_skew"
Boolean **test_check_port_changes_in_capture** = true
float **test_default_bidir_delay** = 5.000000
float **test_default_delay** = 5.000000
float **test_default_period** = 100.000000
string **test_default_scan_style** = "multiplexed_flip_flop"
float **test_default_strobe** = 95.000000
float **test_default_strobe_width** = 0.000000
string **test_infer_slave_clock_pulse_after_capture** = "infer"
Boolean **test_mbc_memory_driven_spec** = "false"
string **test_protocol_add_cycle** = true
Boolean **test_setup_additional_clock_pulse** = "false"
Boolean **test_dft_drc_ungate_clocks** = "false"
Boolean **test_stil_multiclock_capture_procedures** = "false"
string **test_stil_netlist_format** = "db"
Boolean **test_write_four_cycle_stil_protocol** "false"

DESCRIPTION

These variables directly affect the **set_test_methodology**, **read_test_protocol**, **check_test**, **create_test_patterns**, **insert_dft**, and **write_test** commands.

For a list of these variables and their current values, type **print_variable_group test**. To view this manual page online, type **help test_variables**. To view an individual variable description, type **help var**, where var is the name of the variable.

test_allow_clock_reconvergence

When *true* (the default), **check_test** allows reconvergent nets that originate from the same top-level clock port; that is, it does not generate an X when multiple edges reconverge at a net, unless the edges actually conflict. When *false*, **check_test** automatically assigns an X (unknown) whenever multiple edges meet at a gate or net.

test_capture_clock_skew

Allows the user to specify a qualitative measure of clock skew. Values are **no_skew**, **small_skew** (the default), or **large_skew**. For definitions of these values, refer to the **test_capture_clock_skew** manual page.
Because **check_test** does not take into account the delays through capture paths or the skew between clocks, DFT Compiler may inadvertently merge capture groups that should remain disjointed, resulting in bad generated patterns. The variable **test_capture_clock_skew** provides coarse control over the inference process by letting the user specify the amount of clock skew to be assumed.

test_check_port_changes_in_capture

When *true* (the default), **check_test** checks for changes in values applied to bidirectional ports in the parallel measure cycle, which can cause an ureliable capture in sequential cells. When *false*, these checks are disabled.

test_default_bidir_delay
: A positive real number in nanoseconds, which defines the default switching time of bidirectional ports in a tester cycle. This variable is used by both ATPG and TestSim. For ATPG, **test_default_bidir_delay** defines for the design under test the default time at which values are applied (driven) to the bidirectional ports in input mode during parallel measure cycle, and the time at which bidirectional ports are released (undriven) during capture cycle. The value of **test_default_bidir_delay** must be less than the output strobe time, and greater than the capture clock active edge value. For TestSim, **test_default_bidir_delay** defines the default input delay for bidirectional ports in input mode. You should set this variable to the value that was actually used to generate the vectors being input to TestSim; however, the value of this variable must be less than the test clock period.

test_default_delay
: A positive real number in nanoseconds, which defines the default time in a tester cycle to apply values to input ports. This variable is used by both ATPG and TestSim. For ATPG, **test_default_delay** defines for the design under test the default time at which values are applied (driven) to the primary inputs. The value of **test_default_delay** must be less than the output strobe time, and less than the capture clock edge value. For TestSim, **test_default_delay** defines the default input delay for primary inputs. You should set this variable to the value that was actually used to generate the vectors being input to TestSim; however, the value of this variable must be less than the clock period.

test_default_period
: A nonzero value, in nanoseconds, that defines the default length of a test vector cycle. This value translates directly to the speed of application of the test vectors on ATE.

test_default_scan_style
: Defines the default **insert_dft** scan style, to use if a scan style is not specified using **set_scan_style**. The variable must identify a supported scan style; currently one of *multiplexed_flip_flop, clocked_scan, lssd, aux_clock_lssd, combinational*, or *none*. The default is *multiplexed_flip_flop*.

test_default_strobe
: A positive real number in nanoseconds, which defines the default strobe time in a tester cycle for output and bidirectional ports in output mode. This variable is used by both ATPG and TestSim. For ATPG, **test_default_delay** defines for the design under test the default time at which values are strobed on the primary outputs and bidirectional ports in output mode. The value of this variable must be less than or equal to the clock period value. For TestSim, **test_default_strobe** defines the default output strobe time for primary outputs and bidirectional ports acting as outputs. The value of this variable must be less than the clock period.

test_default_strobe_width
: A nonzero value, in nanoseconds, that defines the default strobe pulse width; that is, the default time that specifies how long the strobe pulse needs to be held active after invocation. The sum of this value and the strobe time

value must be less than or equal to the clock period value.

test_infer_slave_clock_pulse_after_capture
: When set to *infer* (the default value), **check_test** protocol inference will be based on an analysis of the scan cell states instead of the scan style as it was previously done (1997.01 and earlier releases). The other values are *pulse* and *no_pulse*. If you set the variable to *pulse*, all slave clocks are pulsed after capture. If you set it to *no_pulse*, no slave clocks are pulsed after capture.

test_mbc_memory_driven_spec
: When set to true, memory-driven spec mode is enabled for MBIST Tech2. This takes the synthesized test specification away from the user and puts it in the hands of the memory model provider.

test_protocol_add_cycle
: When *true* (the default), if you specify design bidirectional ports as inputs during the scan shift, **check_test** adds an extra cycle after the shift cycle in the test protocol. In the extra cycle, all bidirectional ports are set to output mode. If you do not want **check_test** to add the extra cycle, set this variable to **false**.

test_setup_additional_clock_pulse
: When this variable is set to true, an extra clock cycle is added to all clocks in the design in the initialization procedure during execution of the **create_test_protocol** command. The extra cycle ensures that all clock gating latches are at a known state at the end of the initialization procedure.

test_stil_multiclock_capture_procedures
: When *true*, the **write_test_protocol -format stil** command creates capture procedures in the STIL protocol with multiple clocks active in each procedure. The groups of clocks (capture clock groups) are determined by **check_test**. When *false* (the default), **write_test_protocol** places only one clock pulse in each capture clock procedure.

test_stil_netlist_format
: Specifies the netlist format to be used by the **write_test_protocol** command when writing out STIL protocol files. Allowed values are *db* (the default), *verilog*, or *vhdl*.

test_write_four_cycle_stil_protocol
: When *true*, the **write_test_protocol -format stil** command inserts in the output STIL protocol file a dummy cycle between all measure and capture cycles in the STIL protocol. When *false* (the default), no additional cycle is inserted.

SEE ALSO

check_test (2), **create_test_patterns** (2), **insert_dft** (2), **read_test_protocol** (2), **set_min_fault_coverage** (2), **set_test_methodology** (2), **write_test** (2).

test_write_four_cycle_stil_protocol

Instructs the **write_test_protocol -format stil** command to insert in the output STIL protocol file a dummy cycle between all measure and capture cycles in the STIL protocol.

TYPE

Boolean

DEFAULT

false

GROUP

test_variables

DESCRIPTION

Instructs the **write_test_protocol -format stil** command to insert in the output STIL protocol file a dummy cycle between all measure and capture cycles in the STIL protocol. When *false* (the default), no additional cycle is inserted.

The LSI WGL and Verilog vector formats require a dummy cycle to be inserted between the measure and capture cycles so that bidirectional ports can change from output mode to input mode. The **test_write_four_cycle_stil_protocol** variable supports this 4-pattern protocol.

To determine the current value of this variable, type **printvar test_write_four_cycle_stil_protocol**. For a list of test variables and their current values, type **print_variable_group test**.

SEE ALSO

write_test_protocol(2)
test_variables(3)

text_attributes

Contains attributes related to text.

DESCRIPTION

Contains attributes related to text.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified object. Specified with **list_attribute -class text -application**, the definition of attributes can be listed.

Text Attributes

anchor

Specifies the anchor position for text from its **origin**.
The data type of **anchor** is string.
Its valid values are:

- **lb** - Left Bottom
- **cb** - Center Bottom
- **rb** - Right Bottom
- **lc** - Left Center
- **c** - Center Center
- **rc** - Right Center
- **lt** - Left Top
- **ct** - Center Top
- **rt** - Right Top

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox

Specifies the bounding-box of a text. The **bbox** is represented by a **rectangle**. The format of a *rectangle* specification is {{llx lly} {urx ury}}, which specifies the lower-left and upper-right corners of the rectangle.
The data type of **bbox** is string.
This attribute is read-only.

bbox_ll

Specifies the lower-left corner of the bounding-box of a text.
The **bbox_ll** is represented by a **point**. The format of a *point* specification is {x y}.
You can get the **attr_name** of a text, by accessing the first element of its **bbox**.
The data type of **bbox_ll** is string.
This attribute is read-only.

bbox_llx

Specifies x coordinate of the lower-left corner of the bounding-box of a text.
The data type of **bbox_llx** is double.
This attribute is read-only.

bbox_lly

Specifies y coordinate of the lower-left corner of the bounding-box of a text.
The data type of **bbox_lly** is double.
This attribute is read-only.

bbox_ur

Specifies the upper-right corner of the bounding-box of a text.
The fbbox_ur is represented by a **point**. The format of a *point* specification is {x y}.
You can get the **bbox_ur** of a text, by accessing the second element of its **bbox**.
The data type of **bbox_ur** is string.
This attribute is read-only.

bbox_urx

Specifies x coordinate of the upper-right corner of the bounding-box of a text.
The data type of **bbox_urx** is double.
This attribute is read-only.

bbox_ury

Specifies y coordinate of the upper-right corner of the bounding-box of a text.
The data type of **bbox_ury** is double.
This attribute is read-only.

cell_id

Specifies Milkyway design ID in which a text object is located.
The data type of **cell_id** is integer.
This attribute is read-only.

height

Specifies height of a text object.
The data type of **height** is float.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

layer

Specifies layer name of a text object.
The data type of **layer** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

name

Specifies name of a text object.
The data type of **name** is string.
This attribute is read-only.

object_class

Specifies object class name of a text, which is **text**.
The data type of **object_class** is string.
This attribute is read-only.

object_id

Specifies object ID in Milkyway design file.
The data type of **object_id** is integer.
This attribute is read-only.

orientation

Specifies orientation of a text object.
The data type of **orientation** is string.
Its valid values are:

- **N**
- **E**
- **S**
- **W**
- **FN**
- **FE**
- **FS**
- **FW**

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

origin

Specifies origin of a text object.
The **origin** is represented by a **point**. The format of a *point* specification is {x y}.
The data type of **origin** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

text

Specifies the text string to create and display.
The data type of **text** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

SEE ALSO

get_attribute(2),
list_attribute(2),
report_attribute(2),
set_attribute(2).

text_editor_command

Specifies the command that executes when the **Edit/File** menu is selected in the Design Analyzer text window.

TYPE

string

DEFAULT

xterm

GROUP

view_variables

DESCRIPTION

Specifies the command that executes when the **Edit/File** menu is selected in the Design Analyzer text window.

To determine the current value of this variable, type **printvar text_editor_command**. For a list of all **view** variables and their current values, use the **print_variable_group view** command.

SEE ALSO

view_variables (3).

text_print_command

Specifies the command that executes when the **File/Print** menu is selected in the Design Analyzer text window.

TYPE

string

DEFAULT

lpr

GROUP

view_variables

DESCRIPTION

Specifies the command that executes when the **File/Print** menu is selected in the Design Analyzer text window.

To determine the current value of this variable, use **printvar text_print_command**. For a list of all **view** variables and their current values, use the **print_variable_group view** command.

SEE ALSO

view_variables (3).

timing_check_defaults

define the default check list in check_timing command.

TYPE

string

DEFAULT

generated_clock loops no_input_delay unconstrained_endpoints pulse_clock_cell_type no_driving_cell partial_input_delay

DESCRIPTION

This variable defines the default checks to be performed when the check_timing command is executed without any options. The default check list defined by this variable can be overriden by redefining it. The check list modified by using the -override_defaults/-include/-exclude option in check_timing is just valid in one command. Note this variable will not check if the value is correct or not, the check will be done by check_timing command. Each element in check list can be one of the following strings: loops, no_input_delay, unconstrained_endpoints, generated_clock, pulse_clock_cell_type, clock_crossing, data_check_multiple_clock, data_check_no_clock, multiple_clock, generic, gated_clock, ideal_timing, retain, clock_no_period.

EXAMPLE

The following example defines the value of timing_check_defaults variable:

```
prompt> set timing_check_defaults {clock_crossing loops}
```

SEE ALSO

check_timing(2)

timing_clock_gating_propagate_enable

Allow the gating enable signal delay to propagate through the gating cell.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When *true*, the tool allows the delay and slew from the data line of the gating check to propagate. When *false*, the tool blocks the delay and slew from the data line of the gating check from propagating; only the delay and slew from the clock line is propagated.

If the output goes to a clock pin of a latch, setting this variable to *false* produces the most desirable behavior.

If the output goes to a data pin, setting this variable to *true* produces the most desirable behavior.

To determine the current value of this variable, type **printvar timing_clock_gating_propagate_enable** or **echo $timing_clock_gating_propagate_enable**.

SEE ALSO

timing_crpr_remove_clock_to_data_crp

Allows the removal of Clock Reconvergence Pessimism (CRP) from paths that fan out directly from clock source to the data pins of sequential devices.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to *true* then CRP will be removed for all paths that fan out directly from clock source pins to the data pins of sequential devices.

It should be noted that when this variable is set to *true* all sequential devices that reside in the fanout of clock source pins must be handled seperately in the subsequent timing update. This may cause a severe performance degradation to the timing update.

```
      dc_shell-t> echo $timing_crpr_remove_clock_to_data_crp
      or
            dc_shell-t> printvar timing_crpr_remove_clock_to_data_crp
```

SEE ALSO

timing_remove_clock_reconvergence_pessimism(3)

timing_crpr_threshold_ps

Specifies amount of pessimism that clock reconvergence pessimism removal (CRPR) is allowed to leave in the report.

TYPE

float

DEFAULT

20

DESCRIPTION

Specifies amount of pessimism that clock reconvergence pessimism removal (CRPR) is allowed to leave in the report. The unit is in pico seconds (ps), regardless of the units of the main library.

The threshold is per reported slack: setting the this variable to the *TH1* value means that reported slack is no worse than *S - TH1*, where *S* is the reported slack when **timing_crpr_threshold_ps** is set close to zero (the minimum allowed value is **2e-5** picosecond).

The variable has no effect if CRPR is not active (**timing_remove_clock_reconvergence_pessimism** is false). The larger the value of **timing_crpr_threshold_ps**, the faster the runtime when CRPR is active. The recommended setting is about half stage (gate plus net) delay of a typical gate in the clock network. It provides a reasonable trade-off between accuracy and runtime in most cases. You may want to use different settings throughout the design cycle: larger during the design phase, smaller for sign-off. You might have to experiment and set a different value when moving to a different technology.

SEE ALSO

timing_remove_clock_reconvergence_pessimism(3)

timing_disable_cond_default_arcs

Disable the default, non-conditional timing arc between pins that do have conditional arcs.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When *true*, disables nonconditional timing arcs between any pair of pins that have at least one conditional arc. When *false* (the default), these nonconditional timing arcs are not disabled. This variable is primarily intended to deal with the situation between two pins that have conditional arcs, where there is always a default timing arc with no condition.

Set this variable to *true* when the specified conditions cover all possible state-dependent delays, so that the default arc is useless. For example, consider a 2-input XOR gate with inputs as A and B and with output as Z. If the delays between A and Z are specified with 2 arcs with respective conditions ’B’ and ’B~", the default arc between A and Z is useless and should be disabled.

To determine the current value of this variable, type **printvar timing_disable_cond_default_arcs** or **echo $timing_disable_cond_default_arcs**.

SEE ALSO

report_disable_timing(2)

timing_edge_specific_source_latency

Controls whether the generated clock source latency computation will consider edge relationship or not.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to *true*, only the paths with the same sense relationship derived from generated clock definition will be considered. When false, all paths fanout to generated clock source pin will be considered and the worst path will be selected for generated clock source latency computation.

For the current value of this variable, type printvar timing_edge_specific_source_latency.

SEE ALSO

create_generated_clock(2)

timing_enable_multiple_clocks_per_reg

Enables or disables analysis of multiple clocks that reach a single register.

TYPE

Boolean

DEFAULT

false

GROUP

timing_variables

DESCRIPTION

This variable enables or disables analysis of multiple clocks that reach a register clock pin. When true, all clocks reaching the register are costed simultaneously. Since some of the clock interactions might not be present in actual operation, you should use the **set_false_path** command to remove false interactions between mutually exclusive clocks. If there are four or more clocks per register and the design contains level sensitive registers, high impact on runtime might occur. To avoid this, it is recommended that you use the **set_false_path** command to disable false interactions between mutually exclusive clocks or to disable multiple clocks per register.

When false (the default), only one of the clocks reaching the register pin is used.

For the current value of this variable, type the following:

printvar timing_enable_multiple_clocks_per_reg.

SEE ALSO

check_timing(2)
create_clock(2)
create_generated_clock(2)
printvar(2)
set_false_path(2)

timing_enable_non_sequential_checks

Enables or disables library non_sequential checks in the design.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable enables or disables analysis of library non_sequential checks in the design. The non_sequential arcs defined in library will not be used for constraint checking unless this variable is set to true. This variable does not affect the data checks defined by set_data_check command. Enabling the non_sequential checks may cause big delays if the signals reaching the related pin and constrained pin do not belong to the same clock domain. You can use set_multicycle_path command to put appropriate constraints on such paths.

SEE ALSO

set_data_check(2)
set_multicycle_path(2)
printvar(2)

timing_gclock_source_network_num_master_registers

The maximum number of register clock pins clocked by the master clock allowed in generated clock source latency paths.

TYPE

int

DEFAULT

1

DESCRIPTION

This variable allows the user to control the maximum number of register clock pins clocked by the master clock allowed in generated clock source latency paths. The variable does not effect the number of register traversed in a single path that do not have a clock assigned or are clocked by another generated clock that has the same primary master as the generated clock in question.

Register clock pins or transparent-D pins of registers clocked by unrelated clocks are not traversed in determining generated clock source latency paths. An unrelated clock is any clock that primary master clock differs from the generated clock who source latency paths are being computed.

To determine the current value of this variable, type **printvar timing_gclock_source_network_num_master_registers** or **echo $timing_gclock_source_network_num_master_registers**.

SEE ALSO

timing_input_port_clock_shift_one_cycle

Determines whether or not paths originating at input ports are given an extra cycle to meet their timing constraints. This variable will be obsolete in the next release. Please adjust your scripts accordingly.

TYPE

Boolean

DEFAULT

true

GROUP

timing_variables

DESCRIPTION

Affects the behavior of the synthesis timing engine when timing a path from an input port with no clocked input external delay. When *true* (the default value), paths starting at such input ports are given one extra cycle (set_multicycle_path 2) to meet timing constraints at clocked destination registers or output ports. When *false*, no extra multicycle shift is applied.

To determine the current value of this variable, use **printvar timing_input_port_clock_shift_one_cycle**.

SEE ALSO

report_timing(2)

timing_input_port_default_clock

Determines whether a default clock is assumed at input ports for which you have not defined a clock-specific input external delay.

TYPE

Boolean

DEFAULT

true

GROUP

timing_variables

DESCRIPTION

This Boolean variable affects the behavior of the synthesis timing engine when timing a path from an input port with no clocked input external delay. When *true* (the default value), all such input ports are given one imaginary clock so that the inputs are constrained. This also causes the clocks along the paths driven by these input ports to become related. When *false*, no such imaginary clock is assumed.

To determine the current value of this variable use **printvar timing_input_port_default_clock**.

SEE ALSO

report_timing(2)

timing_remove_clock_reconvergence_pessimism

Enables or disables clock reconvergence pessimism removal.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to *true*, the synthesis timing engine removes clock reconvergence pessimism from slack calculation and minimum pulse width checks.

Clock reconvergence pessimism (CRP) is a difference in delay along the common part of the launching and capturing clock paths. The most common causes of CRP are reconvergent paths in the clock network, and different minimum and maximum delay of cells in the clock network.

Any effective change in the value of the **timing_remove_clock_reconvergence_pessimism** variable causes full **update_timing**. You cannot perform one **report_timing** operation that considers CRP and one that does not without full **update_timing** in between.

To run compile with CRP removal, the clock network must be dont_touched.

```
prompt> set timing_remove_clock_reconvergence_pessimism true
true

prompt> report_timing
```

SEE ALSO

report_timing(2)
timing_crpr_threshold_ps(3)

timing_report_attributes

Specifies the list of attributes to be reported with the **report_timing -attributes** command.

Note: This variable will be obsolete in the next release. Please adjust your scripts accordingly.

TYPE

list

DEFAULT

{dont_touch dont_use map_only size_only ideal_net}

GROUP

timing_variables

DESCRIPTION

Specifies the list of attributes to be reported with the **report_timing -attributes** command. Attributes currently supported are **dont_touch**, **dont_use**, **map_only**, **size_only**, and **ideal_net**.

The default value for this attribute is set in the system .synopsys_dc.setup file.

To determine the current value of this variable, type **printvar timing_report_attributes**. For a list of all timing variables and their current values, type **print_variable_group timing**.

SEE ALSO

compile(2)
report_timing(2)
timing_variables(3)

timing_self_loops_no_skew

Affects the behavior, runtime, and CPU usage of **report_timing** and **compile**.

Note: This variable will be obsolete in the next release. Please adjust your scripts accordingly.

TYPE

Boolean

DEFAULT

false

GROUP

timing_variables

DESCRIPTION

Affects the behavior, runtime, and CPU usage of **report_timing** and **compile**. When set to *true*, clock skew is eliminated for a path that starts and ends at the same register. When set to *false* (the default value), clock skew is not eliminated. Thus, the timing for such paths is pessimistic. To obtain the more accurate behavior of no clock skew (uncertainty) for such paths, set this variable to *true*. However, note that runtime and memory usage might increase significantly.

To determine the current value of this variable, type **printvar timing_self_loops_no_skew**. For a list of all **timing** variables and their current values, type **print_variable_group timing**.

SEE ALSO

compile(2)
report_timing(2)
timing_variables(3)

timing_separate_clock_gating_group

Specifies if a separate cost group is used for clock gating checks in timing analysis, reports, and optimization.

TYPE

Boolean

DEFAULT

false

GROUP

timing

DESCRIPTION

When set to true, a separate cost group named "****clock_gating_default****" is created for all the clock gating checks; when false, the clock gating check is applied to the cost group of the clock being gated. The default value is false. If multiple scenarios exist, a cost group is created for clock gating checks for each scenario when this variable is true.

You can change the weight and critical_range settings for this cost group by using the **group_path** command with **-name "**clock_gating_default**"**.

SEE ALSO

get_path_groups(2)
group_path(2)
report_clock_gating(2)
report_clock_gating_check(2)
report_constraint(2)
report_path_group(2)
report_qor(2)
report_timing(2)

timing_use_clock_specific_transition

Propagate the transition from the specific clock path for latency calculation

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When set to true (the default value), ICC only propagates the transition of the clock path, for the purpose of clock latency calculation. If there are multi input gates on the clock network, the transition of non-clock inputs are ignored during clock latency calculation.

For generated clocks defined on an output of a gate, ICC propagates the transition from the path connected to its master clock, and uses that transition value for the purpose of calculation clock latency for the generated clock.

When set to false, ICC will allow transition from the non-clock input of multi-input gates along the clock path to be used for clock latency calculation

In addition, when set to false, generated clock defined at the output of a gate will use zero transition at the generated clock source.

To determine the current value of this variable, type **printvar timing_use_clock_specific_transition** or **echo $timing_use_clock_specific_transition**.

SEE ALSO

timing_use_driver_arc_transition_at_clock_source

Uses the backward cell arc to compute a realistic driver model at the driver pin for primary clock sources and also generated clock that can not trace back to its master clock

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When set to true (the default value), ICC use the backward cell arcs, when at least one does exist, to compute a worst-case driver model. This behavioral applies to the primary clock sources, which are defined by create_clock, and generated clock sources (defined by create_generated_clock) that can not trace back to its master clock.

When set to false, ICC asserts a zero-transition, ideal ramp model at the driver pin.

To determine the current value of this variable, type **printvar timing_use_driver_arc_transition_at_clock_source** or **echo $timing_use_driver_arc_transition_at_clock_source**.

SEE ALSO

timing_use_enhanced_capacitance_modeling

Specifies the use of the following attributes found on a library pin: rise_capacitance_range(low, high), fall_capacitance_range(low, high), rise_capacitance, and fall_capacitance.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this environment variable is set to true, different capacitance values for rise and fall will be used in delay calculation, if such values are specified in the library. If no such attributes are found, it will use the default capacitance attribute.

With the bc/wc analysis type, the (low, high) values from the worst condition will be used for the setup analysis, and the (low, high) value from the best condition will be used for the hold analysis.

If the analysis type is on_chip_variation, the high value from the max condition will be used in computing the late arrivals, and at the same time the low value from the min condition will be used in the early arrivals.

SEE ALSO

set_operating_conditions(2)

timing_variables

Variables that affect timing.

SYNTAX

Boolean **create_clock_no_input_delay** = false
Boolean **disable_library_transition_degradation** = false
Boolean **disable_auto_time_borrow** = false
Boolean **timing_self_loops_no_skew** = false
int **true_delay_prove_false_backtrack_limit** = 1000
int **true_delay_prove_true_backtrack_limit** = 1000
list **timing_report_attributes** = {"dont_touch" "dont_use" "map_only" "size_only"}
Boolean **disable_case_analysis** = false
Boolean **case_analysis_with_logic_constants** = false
String **case_analysis_log_file** = ""
Boolean **enable_slew_degradation** = "false"
Boolean **high_fanout_net_threshold** = "1000"

DESCRIPTION

These variables directly affect timing aspects in **dc_shell**. Defaults are shown under Syntax.

For a list of timing_variables, type **print_variable_group timing**. To view this manual page online, type **help timing_variables**. To view an individual variable description, type **help *var***, where *var* is the name of the variable.

create_clock_no_input_delay

Affects delay propagation characteristics of clock sources created using **create_clock**. When **create_clock_no_input_delay** is *false* (the default value), clock sources used in the data path are established as timing startpoints. The clock sources in the design will propagate rising delays on every rising clock edge, and will propagate falling delays on every falling clock edge. You can disable this behavior by setting **create_clock_no_input_delay** to *true*.

disable_auto_time_borrow

Affects automatic time borrowing. When **disable_auto_time_borrow** is *false* (the default value), automatic time borrowing balances the slack along back-to-back latch paths to reduce the overall delay cost. When *true*, no slack balancing will occur during time borrowing.

disable_library_transition_degradation

When *false* (the default), **report_timing** and other commands that use timing in dc_shell use the transition degradation table in the library to determine the net transition time. When *true*, the timing commands behave as though there were no transition degradation across the net.

disable_library_transition_degradation

When *false* (the default), **report_timing** and other commands that use timing in dc_shell try to balance the slack between paths. When *true*, the cycle time is portioned such that the first path attempts to make timing and the second path only gets the remainder of the delay.

timing_self_loops_no_skew

This Boolean variable affects the behavior, runtime, and cpu usage of **report_timing** and **compile**. When *true*, register. When *false* (the default value), clock skew is not eliminated; thus, the timing for such paths is pessimistic. To obtain the more accurate behavior of no clock skew (uncertainty) for such paths, set the variable to *true*, but note that runtime and memory usage may increase significantly.

true_delay_prove_false_backtrack_limit

An integer variable that specifies the number of backtracks to be used by **report_timing -true** in searching for false paths; **-1** specifies unlimited backtracking. The default is 1000. One of a pair of variables that includes **true_delay_prove_true_backtrack_limit**.

true_delay_prove_true_backtrack_limit

An integer variable that specifies the number of backtracks to be used by **report_timing -true** in searching for true paths; **-1** specifies unlimited backtracking. The default is 1000. One of a pair of variables that includes **true_delay_prove_false_backtrack_limit**.

timing_report_attributes

Specifies the list of attributes to be reported with **report_timing -attributes**. The current list of attributes supported are dont_touch, dont_use, map_only, and size_only.

disable_case_analysis

Determines whether constant propagation is performed in the design from pins either that are tied to a logic constant value, or for which a **case_analysis** command is specified. By default, constant propagation happens if the **set_case_analysis** is specified or if the variable case_analysis_with_logic_constants is set to true, unless the variable disable_case_analysis is set to false.

case_analysis_with_logic_constants

Determines whether constant propagation will be preformed if there are only logic constants in the circuit and there is no set_case_analysis command used. By default, logic constants are not propagated in the absense of set_case_analysis commands or case_analysis_with_logic_constants being set to true.

case_analysis_log_file

Specifies the name of a log file to be generated during propagation of constant values from case analysis or from nets tied to logic zero or to logic one. The log file contains the list of all ports and pins that propagate constants.

enable_slew_degradation

When *false* (the default setting), optimization with physical information does not consider transition degradation across a net and only tries to fix transition violations on a per net basiss. Setting this variable to *true* will cause optimization with physical information to consider transition degradation across a net and try to fix transition violations on a per cell basis.

high_fanout_net_threshold

Specifies the minimum number of loads for a net to be classified as a high-fanout net, the default is 1000. The delays and loads of high-fanout are computed using a simplified model assuming a fixed fanout number. The rationale behind this is that delays of high-fanout nets are expensive to compute but such nets are often unconstrained (as in the case of global reset nets, scan enable nets, and so on). So detailed delay calculations on such nets are expensive and usually unnecessary.

timing_disable_internal_inout_net_arcs

This variable is used to disable bidirectional feedback paths that involve more than one cell. No path segmentation is required. By default this variable is true. If the variable is set false, then bidirectional feedback loop would be enabled.

timing_disable_internal_inout_cell_paths

This variable is used to disable bidirectional feedback paths that are present within a cell. By default, the variable is true. If the variable is set false, then bidirectional feedback path would be present.

SEE ALSO

compile (2), **create_clock(2), report_timing** (2); **create_clock_no_input_delay(3), disable_library_transition_degradation(3), timing_self_loops_no_skew(3), true_delay_prove_false_backtrack_limit(3), true_delay_prove_true_backtrack_limit(3), timing_report_attributes(3), disable_case_analysis(3), case_analysis_with_logic_constants(3), case_analysis_log_file(3), enable_slew_degradation(3), high_fanout_net_threshold(3).**

track_attributes

Contains attributes related to track.

DESCRIPTION

Contains attributes related to track.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified track. Specified with **list_attribute -class track -application**, the definition of attributes can be listed.

Track Attributes

bbox

Specifies the bounding-box of a track. The **bbox** is represented by a **rectangle**. The format of a *rectangle* specification is {{llx lly} {urx ury}}, which specifies the lower-left and upper-right corners of the rectangle.
The *bbox* of a track is calculated by the **origin** and **orientation** of its cell and the actual **bbox** of its corresponding terminal from the child MW design.
This attribute is read-only.

bbox_ll

Specifies the lower-left corner of the bounding-box of a track.
The **bbox_ll** is represented by a **point**. The format of a *point* specification is {x y}.
You can get the **bbox_ll** of a track by accessing the first element of its **bbox**.
This attribute is read-only.

bbox_llx

Specifies x coordinate of the lower-left corner of the bounding-box of a track.
The data type of **bbox_llx** is double.
This attribute is read-only.

bbox_lly

Specifies y coordinate of the lower-left corner of the bounding-box of a track.
The data type of **bbox_lly** is double.
This attribute is read-only.

bbox_ur

Specifies the upper-right corner of the bounding-box of a track.
The **bbox_ur** is represented by a **point**. The format of a *point* specification is {x y}.
You can get the **bbox_ur** of a track by accessing the second element of its **bbox**.
This attribute is read-only.

bbox_urx

Specifies x coordinate of the upper-right corner of the bounding-box of a track.

The data type of **bbox_urx** is double.
This attribute is read-only.

bbox_ury
Specifies y coordinate of the upper-right corner of the bounding-box of a track.
The data type of **bbox_ury** is double.
This attribute is read-only.

cell_id
Specifies Milkyway design ID in which a track object is located.
This attribute is read-only.

count
Specifies the number of grid of a track.
The data type of **count** is integer.
This attribute is read-only.

direction
Specifies the routing direction of a track.
The valid values can be: X, Y.
This attribute is read-only.

layer
Specifies the layer name where a track object is located on.
This attribute is read-only.

name
Specifies the object name of a track.
This attribute is read-only.

object_class
Specifies object class name of a track, which is **track**.
This attribute is read-only.

object_id
Specifies object ID in Milkyway design file.
This attribute is read-only.

space
Specifies track step of a track.
The data type of **space** is integer.
This attribute is read-only.

start
Specifies start position of a track.
The data type of **start** is coordinate point.
This attribute is read-only.

stop
Specifies end position of a track.
The data type of **stop** is coordinate point.
This attribute is read-only.

SEE ALSO

```
get_attribute(2),
list_attribute(2),
report_attribute(2).
```

ungroup_keep_original_design

Controls ungroup and compile command to keep the original design when a design is ungrouped in XG mode.

TYPE

Boolean

DEFAULT

false

GROUP

none

DESCRIPTION

By default, **ungroup** and **compile** in XG, will delete the original design if all the instances of a design has be ungrouped and there are no other references to the design from anywhere else.

When set to true, the original design is preserved.

For example, if are two instances of design mid named mid1 and mid2 and **ungroup -flatten -all** is issued, after ungroup collapses the hierarchies, the design called mid is deleted from memory if there are no other references to the design from elsewhere. This variable is used to change the behaviour. When this variable is set to true explicitly, the ungrouped design will be preserved.

SEE ALSO

ungroup(2)
compile(2)
set_ungroup(2)

uniquify_keep_original_design

Controls uniquify command to keep the original design when a multiply instantiated design is uniqufied in XG mode.

TYPE

Boolean

DEFAULT

false

GROUP

none

DESCRIPTION

By default, **uniquify** in XG, will delete the original design if all the instances of a design has been uniqufiied and there are no other references to the design from anywhere else.

When set to true, the original design is preserved.

For example, if are two instances of design mid, uniquify creates two new designs mid_1 and mid_2. By default the original design "mid" is deleted in XG. This variable is used to change the behaviour.

SEE ALSO

uniquify(2)

uniquify_naming_style

Specifies the naming convention to be used by the **uniquify** command.

TYPE

string

DEFAULT

GROUP

system_variables

DESCRIPTION

Specifies the naming convention to be used by the **uniquify** command. The variable string must contain only one %s (percent s) and one %d (percent d) character sequence. To use a percent sign in the design name, two are needed in the string (%%).

To determine the current value of this variable, type **printvar uniquify_naming_style**. For a list of all **system** variables and their current values, use the **print_variable_group system** command.

SEE ALSO

uniquify(2)
system_variables(3)

upf_extension

Sets the variable to false to disable writing of UPF extension commands in save_upf.

TYPE

Boolean

DEFAULT

true

GROUP

mv

DESCRIPTION

This variable controls whether the **save_upf** command writes UPF extension commands such as **set_related_supply_net** into UPF files. By default, UPF extension commands are written by **save_upf** in the following format:

```
   if {[info exists upf_extension] && upf_extension} {
   <upf_extension_command>
   }
```

If **upf_extension** is set to false, these lines are not written by **save_upf**. The UPF extension command itself continues to be written by **write_script**, and continues to be readable by the **source** and **load_upf** commands.

All tools, including third party tools, that need to support the UPF extension commands must predefine the **upf_extension** Tcl variable as true.

SEE ALSO

set_related_supply_net(2)

use_port_name_for_oscs

Specifies that when off-sheet connectors for nets also have ports on them, they are given the name of the port.

TYPE

Boolean

DEFAULT

false

GROUP

schematic_variables

DESCRIPTION

Specifies that when off-sheet connectors for nets also have ports on them, they are given the name of the port. The value of *true* is the default for this partitioning option. When the value is set to *false*, the connectors are given the name of the net.

To determine the current value of this variable, type **printvar use_port_name_for_oscs**. For a list of all **schematic** variables and their current values, use the **print_variable_group schematic** command.

SEE ALSO

schematic_variables(3)

verbose_messages

Causes more explicit system messages to be displayed during the current Design Analyzer dc_shell session.

TYPE

Boolean

DEFAULT

true

GROUP

system_variables

DESCRIPTION

Causes more explicit system messages to be displayed during the current Design Analyzer dc_shell session. The default is *true*.

To determine the current value of this variable, type **printvar verbose_messages**. For a list of all **system** variables and their current values, type **print_variable_group system**.

SEE ALSO

system_variables(3)

verilogout_equation

Writes Verilog "assign" statements (Boolean equations) for combinational gates, rather than gate instantiations.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

Writes Verilog "assign" statements (Boolean equations) for combinational gates, rather than gate instantiations. The default value is *false*.

To determine the current value of this variable, use **printvar verilogout_equation**. For a list of all **hdl** variables and their current values, use the **print_variable_group hdl** command.

SEE ALSO

hdl_variables(3)

verilogout_higher_designs_first

Writes Verilog "modules" so that the higher level designs come before lower level designs, as defined by the design hierarchy.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

Writes Verilog "modules" so that higher level designs come before lower level designs, as defined by the design hierarchy. The default is to write lower level designs first.

To determine the current value of this variable, use **printvar verilogout_higher_designs_first**. For a list of all **hdl** variables and their current values, use the **print_variable_group hdl** command.

SEE ALSO

hdl_variables(3)

verilogout_ignore_case

Instructs the compiler not to consider case when comparing identifiers to Verilog reserved words.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

Instructs the compiler not to consider case when comparing identifiers to Verilog reserved words.

When an identifier is equal to a reserved word, the identifier is "escaped" by putting a backslash ('\') in front of it. When this variable is set to *false*, case is considered. When set to *true*, case is ignored in the comparison.

Therefore, if **verilogout_ignore_case** is set to *true*, the default allows an identifier BUF to pass unchanged, BUF becoming \BUF.

To determine the current value of this variable, type **printvar verilogout_ignore_case**. For a list of all **hdl** variables and their current values, type **print_variable_group hdl**.

SEE ALSO

hdl_variables(3)

verilogout_include_files

Specifies to the **write -f verilog** command to write an include statement that will have the name of the value you set for this variable.

TYPE

list

DEFAULT

""

GROUP

hdl_variables

DESCRIPTION

Specifies to the **write -f verilog** command to write an include statement that will have the name of the value you set for this variable. For example, when **verilogout_include_files**={"my_header.v"}, you see an include "my_header.v" in your Verilog output.

To determine the current value of this variable, type **printvar verilogout_include_files**. For a list of all **hdl** variables and their current values, type **print_variable_group hdl**.

SEE ALSO

write(2)
hdl_variables(3)

verilogout_no_tri

Declares three-state nets as Verilog "wire" instead of "tri." This variable is useful in eliminating "assign" primitives and "tran" gates in the Verilog output.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

Declares three-state nets as Verilog "wire" instead of "tri." This variable is useful in eliminating "assign" primitives and "tran" gates in the Verilog output.

To determine the current value of this variable, use **printvar verilogout_no_tri**. For a list of all **hdl** variables and their current values, use the **print_variable_group hdl** command.

SEE ALSO

hdl_variables(3)

verilogout_show_unconnected_pins

Instructs the Verilog writer in dc_shell to write out all of the unconnected instance pins, when connecting module ports by name. For example, modb b1 (.A(in),.Q(out),.Qn()).

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

Instructs the Verilog writer in dc_shell to write out all of the unconnected instance pins, when connecting module ports by name. For example, modb b1 (.A(in),.Q(out),.Qn()).

When this variable is set to *false* (the default), the Verilog writer does not write out any unconnected pins. For example, modb b1 (.A(in),.Q(out)).

To determine the current value of this variable, use **printvar verilogout_show_unconnected_pins**. For a list of all **hdl** variables and their current values, use the **print_variable_group hdl** command.

SEE ALSO

hdl_variables(3)

verilogout_single_bit

Instructs the compiler not to output vectored ports in the Verilog output. All vectors are written as single bits.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

Instructs the compiler not to output vectored ports in the Verilog output. All vectors are written as single bits.

To determine the current value of this variable, use **printvar verilogout_single_bit**. For a list of all **hdl** variables and their current values, use the **print_variable_group hdl** command.

SEE ALSO

hdl_variables(3)

verilogout_unconnected_prefix

Instructs the Verilog writer in dc_shell to use the name SYNOPSYS_UNCONNECTED_ to create unconnected wire names. The general form of the name is SYNOPSYS_UNCONNECTED_%d.

TYPE

string

DEFAULT

SYNOPSYS_UNCONNECTED_

GROUP

hdl_variables

DESCRIPTION

Instructs the Verilog writer in dc_shell to use the name (SYNOPSYS_UNCONNECTED_) to create unconnected wire names. The general form of the name is SYNOPSYS_UNCONNECTED_%d.

The purpose of this variable is to avoid conflict with the name already used in the design.

To determine the current value of this variable, use **printvar verilogout_unconnected_prefix**. For a list of all **hdl** variables and their current values, use the **print_variable_group hdl** command.

SEE ALSO

hdl_variables(3)

vhdlio_variables

SYNTAX

```
string vhdllib_architecture = "VITAL"
Boolean vhdllib_glitch_handle = "true"
Boolean vhdllib_logic_system = "ieee-1164"
Boolean vhdllib_logical_name = ""
Boolean vhdllib_negative_constraint = "false"
string vhdllib_pulse_handle =   "use vhdllib_glitch_handle"
Boolean vhdllib_sdf_edge = "false"
Boolean vhdllib_tb_compare = "0"
Boolean vhdllib_tb_x_eq_dontcare = "false"
Boolean vhdllib_timing_checks = "true"
Boolean vhdllib_timing_mesg = "true"
Boolean vhdllib_timing_xgen = "false"
Boolean vhdllib_vital_99 = "false"
string vhdlout_architecture_name = "SYN_%a_%u"
string vhdlout_bit_type = "std_logic"
Boolean vhdlout_bit_type_resolved =   "true"
string vhdlout_bit_vector_type = "std_logic_vector"
list vhdlout_conversion_functions = {}
Boolean vhdlout_dont_create_dummy_nets = "false"
Boolean vhdlout_dont_write_types = "FALSE"
Boolean vhdlout_equations = "FALSE"
Boolean vhdlout_follow_vector_directions = "true"
Boolean vhdlout_lower_design_vector = "TRUE"
string vhdlout_one_name = "`1'"
string vhdlout_package_naming_style = "CONV_PACK_%d"
string vhdlout_preserve_hierarchical_types = "VECTOR"
Boolean vhdlout_separate_scan_in =   "FALSE"
string vhdlout_single_bit = "USER"
Boolean vhdlout_synthesis_off = "TRUE"
string vhdlout_target_simulator = ""
string vhdlout_three_state_name = "`Z'"
string vhdlout_three_state_res_func = ""
float vhdlout_time_scale =   "1.0"
string vhdlout_top_configuration_arch_name = "A"
string vhdlout_top_configuration_entity_name = "E"
string vhdlout_top_configuration_name = "CFG_TB_E"
string vhdlout_unknown_name = "'X'"
list vhdlout_use_packages = "IEEE.std_logic_1164"
string vhdlout_wired_and_res_func = ""
string vhdlout_wired_or_res_func = ""
Boolean vhdlout_write_architecture = "TRUE"
Boolean vhdlout_write_components = "TRUE"
Boolean vhdlout_write_entity = "TRUE"
Boolean vhdlout_write_top_configuration = "FALSE"
string vhdlout_zero_name = "`0'"
string write_test_vhdlout = "textio"
```

DESCRIPTION

Variables that directly affect the **write**, **write_lib**, and **write_test** commands for VHDL format.

The VHDL library variables (**vhdllib**) determine the timing violation behavior (for example, hazards and forbidden timing events) of the generated VHDL library. For more information on VHDL library variables, refer to the *Library Compiler Reference Manual: VHDL Libraries*.

The VHDL output variables (**vhdlout**) determine the format of the VHDL design output file. For more information on VHDL output variables, refer to the *VHDL Compiler Reference Manual: VHDL Libraries* and the **hdl_variables** and **io_variables** manual pages.

To view this manual page online, type **man vhdlio_variables**. To view an individual variable description, type **man *var_name***.

vhdllib_architecture
    Determines the VHDL model types for the **write_lib** command to generate. **VITAL** selects VITAL models.

vhdllib_glitch_handle
    When **true** (the default value), timing hazards have glitch-forced (glitch on detect) **X**s. When **false**, **X**s are spike-forced (glitch on event).

vhdllib_logic_system
    Selects the logic system in which to create the VHDL libraries. The default value (ieee-1164) is the only valid choice available. It represents the IEEE Std 1164.1 logic system. Do not change the value of this variable to anything other than ieee-1164.

vhdllib_logical_name
    Defines the logical name to be used by the VHD libraries.

vhdllib_negative_constraint
    When **true**, all cells in the generated VITAL library can handle negative timing constraints, ignoring the cell attribute **handle_negative_constraint** value set in the technology library. When **false** (the default), only cells in the generated VITAL library whose cell attribute **handle_negative_constraint** is **true** can handle negative timing constraints.

vhdllib_pulse_handle
    Specifies whether timing hazards are to have glitch_forced (glitch on detect) or spike_forced (glitch on event) Xs. Default value is **true**.

vhdllib_sdf_edge
    Determines whether edge information is to be added to delay generics in VITAL libraries. The default value is **false**.

vhdllib_tb_compare
    Controls library testbench generation. No testbenches are created if this variable is set to **0** (the default). Otherwise, **vhdllib_tb_compare** specifies the default value for the testbench's **Cmp_Algorithm** generic integer flag. The greater the number from 1 to 5, the more rigorous the verification.

vhdllib_tb_x_eq_dontcare

Specifies the default value for the **X_Eq_DontCare** generic Boolean flag of the testbenches. If **X_Eq_DontCare** is **true**, X states are ignored during output comparisons. The default is **false**.

vhdllib_timing_checks

Determines the default value of the cell **TimingChecksOn** generic Boolean flag in VITAL model. The default value is **true**.

vhdllib_timing_mesg

Determines the value of GlitchMode parameter for VitalPropagatePathDelay procedure in VITAL model. The default is *true*.

vhdllib_timing_xgen

Determines the default value for the **XGenerationOn** generic Boolean flag in the VITAL model. The default is *false*.

vhdllib_vital_99

Determines whether VITAL libraries are to incorporate changes for VITAL 99. The default is **false**.

vhdlout_architecture_name

Determines the name that will be used for the architecture the **write -f vhdl** command writes out.

vhdlout_bit_type

Sets the basic bit type in a design written to VHDL. This is useful when your design methodology is based on a logic value system that is not std_logic. The default is *std_logic*.

vhdlout_bit_type_resolved

When **true** (the default value), prevents **VHDLout** from creating new bus resolution functions when writing wired logic. When **false**, **vhdlout** creates bus resolution functions for signals with more than one driver. Set this variable to **true** if the **vhdlout_bit_type** is a resolved type.

vhdlout_bit_vector_type

Sets the basic bit_vector type in a design written to VHDL. This is useful when your design methodology is based on a logic value system that is not std_logic.

vhdlout_conversion_functions

Overrides conversion functions that are written out. The value of this variable is a list of lists. Each lower level list is a list of three strings representing the "from type," the "to type," and the function used for the conversion.

vhdlout_dont_create_dummy_nets

When *true*, the vhdl writer does not create any dummy nets for connecting unused pins or ports.

vhdlout_dont_write_types

When *true*, and if **vhdlout_single_bit** is *false* (the default value), type declarations for any types that are declared in the original VHDL are not written. Only special types needed by **vhdlout** are declared, and are given unique names. Other types should be declared in a user-defined package using **vhdlout_use_packages**.

vhdlout_equations
: When *true*, combinational logic is written as technology-independent Boolean equations, and sequential logic is written as technology-independent wait statements. When *false* (the default value), all logic is written as technology-specific netlists.

vhdlout_follow_vector_direction
: When **true** (the default value), the VHDL writer writes out correct array range direction. For example, 10 down to 0, if the original is 10 down to 0.

vhdlout_lower_design_vector
: Determines the way in which the **write -f vhdl** command writes out ports on lower-level designs. The default value is **true**.

vhdlout_one_name
: Determines the literal name for constant bit value ‘1’ in a design written to VHDL. This is useful when your design methodology is based on a more general logic value than "BIT". This variable is used with **vhdlout_bit_type**.

vhdlout_package_naming_style
: Determines the name used for the type conversion packages written out by vhdlout.

vhdlout_preserve_hierarchical_types
: Affects how ports on lower-level designs are written out with **write -f vhdl**. A lower-level design is instantiated by any of the designs being written out. In contrast, ports on top-level designs are controlled by **vhdlout_single_bit**. Allowed values are *USER*, *VECTOR* (the default), and *BIT*. For definitions of the allowed values, refer to the **vhdlout_preserve_hierarchical_types** manual page.

vhdlout_separate_scan_in
: Affects how the scan chain is written out in VHDL. When **true**, the scan chain is written out as a separate file from the design; when *false* (the default value), the scan chain is written out in the same file as the design.

vhdlout_single_bit
: Affects how ports on the top-level design are written out. Lower level design ports are controlled by **vhdlout_preserve_hierarchical_types**. A design is considered lower level if it is instantiated by any of the designs being written out.

vhdlout_synthesis_off
: This variable has an effect only if **vhdlout_write_top_configuration** is **true**. When **vhdlout_synthesis_off** is **true** (the default), if **vhdlout_write_top_configuration** is also **true**, **write -format vhdl** writes out "synthesis_off" and "synthesis_on" at the beginning and at the end of the configuration statement. When **vhdlout_synthesis_off** is *false*, **write -format vhdl** does not write out these strings.

vhdlout_target_simulator
: Names the target simulator to which the VHDL file is written. Currently, the

only valid value is *xp*.

vhdlout_three_state_name
: Names the high impedance bit value used for three-state device values. The default is *Z*.

vhdlout_three_state_res_func
: Names a user-supplied three-state resolution function, which must be in one of the packages specified by **vhdlout_use_packages**. If this variable is set to an empty string, a default three-state resolution function is written out, if needed. The default three-state resolution function drives a signal to "unknown" if the signal is driven more than once by logic zero or logic one.

vhdlout_time_scale
: Specifies the scaling of the delays the tool writes to timing files, in Synopsys VHDL format. The **write_timing** command writes delays from the tool to timing files.

vhdlout_top_configuration_arch_name
: Determines the name of the outside architecture. Depending on the setting of **vhdlout_write_top_configuration**, a configuration statement is written out by vhdlout.

vhdlout_top_configuration_entity_name
: Determines the name of the outside entity. Depending on the setting of **vhdlout_write_top_configuration**, a configuration statement is written by vhdlout.

vhdlout_top_configuration_name
: Determines the name of the configuration that is written out under control of **vhdlout_write_top_configuration**.

vhdlout_top_design_vector
: Affects the way in which the **write -f vhdl** command writes out ports on the top-level design.

vhdlout_unconnected_pin_prefix
: Affects the way in which the **write -f vhdl** command writes out unconnected pin names.

vhdlout_unknown_name
: Specifies the value used to drive a signal to the unknown state. The default for this variable is *X*.

vhdlout_use_packages
: A list of package names. A "use" clause is written into the VHDL file for each of these packages for all entities. If this variable is not set, or is set to an empty list ({}), it has no effect on the **write** command.

vhdlout_wired_and_res_func
: Specifies the name of a wired AND resolution function. This user-supplied function must be in one of the packages specified by **vhdlout_use_packages**. If this variable is set to an empty string, a default wired AND resolution function is written out, if needed.

vhdlout_wired_or_res_func
    Specifies the name of a wired OR resolution function. This user-supplied function must be in one of the packages specified by **vhdlout_use_packages**. If this variable is set to an empty string, a default wired OR resolution function is written out, if needed.

vhdlout_write_architecture
    When **true** (the default value), **vhdlout** writes architecture declarations. When *false*, no architecture declarations are written. This variable may be used with **vhdlout_write_entity** and **vhdlout_write_top_configuration** to control what information will be written out by the **write -format vhdl** command.

vhdlout_write_components
    When **true** (the default value), **vhdlout** writes component declarations for cells mapped to a technology library. When *false*, no component declarations are written. Component declarations are required by VHDL. If you set this variable to *false*, make sure that **vhdlout_use_packages** includes a package containing the necessary component declarations.

vhdlout_write_entity
    When **true** (the default value), **vhdlout** writes entity declarations. When *false*, no entity declarations are written. This variable may be used with **vhdlout_write_architecture** and **vhdlout_write_top_configuration** to control the information that will be written out by the **write -format vhdl** command.

vhdlout_write_top_configuration
    When **true**, **vhdlout** writes a configuration statement, if necessary. It is necessary when ports on the topmost design are written as vectors instead of user types. When *false*, no configuration statement is written.

vhdlout_zero_name
    Determines the literal name for constant bit value '0' in a design written to VHDL. This is useful when your design methodology is based on a more general logic value than BIT. Used with **vhdlout_bit_type**. The default is *0*.

write_test_vhdlout
    Determines whether the **write_test -format vhdl** command generates a VHDL test program in TEXTIO format.

SEE ALSO

write(2)
write_lib(2)
write_test(2)
hdl_variables(3)
io_variables(3)

vhdllib_architecture

Determines the VHDL model types for the **write_lib** command to generate.

TYPE

string

DEFAULT

VITAL

GROUP

vhdlio_variables

DESCRIPTION

The **vhdllib_architecture** variable specifies the VHDL model type the **write_lib** command is to generate.

If you set the value of of this variable to VITAL (the default), the tool generates VITAL simulation models.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdllib_architecture
```

For a list of all **vhdllib** variables and their current values, see the **vhdlio_variables.3** man page.

SEE ALSO

vhdlio_variables(3)

vhdllib_glitch_handle

Specifies whether timing hazards are to have glitch-forced (glitch on detect) or spike-forced (glitch on event) Xs.

TYPE

Boolean

DEFAULT

true

GROUP

vhdlio_variables

DESCRIPTION

Use the **vhdllib_glitch_handle** variable to specify whether timing hazards are to have Xs that are glitch-forced (glitch on detect) or spike-forced (glitch on event).

If you set the value of this variable to true (the default value), Xs are to be glitch-forced. If you set the value to false, Xs are to be spike-forced.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdllib_glitch_handle
```

For a list of all **vhdllib** variables and their current values, see the **vhdlio_variables.3** man page.

SEE ALSO

vhdlio_variables(3)
vhdllib_pulse_handle(3)

vhdllib_logic_system

Specifies the logic system in which the tool is to create the VHDL libraries.

TYPE

Boolean

DEFAULT

ieee-1164

GROUP

vhdlio_variables

DESCRIPTION

Use the **vhdllib_logic_system** variable to specify the logic system in which the tool is to create the VHDL libraries.

The default value (ieee-1164) of this variable is the only choice available. It represents the IEEE Std 1164.1 logic system.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdllib_logic_system
```

For a list of all **vhdllib** variables and their current values, see the **vhdlio_variables.3** man page.

SEE ALSO

vhdlio_variables(3)

vhdllib_logical_name

This variable defines the logical name to be used by the VHDL libraries. If the variable is set to an empty string (the default), the file base name is used as the default.

TYPE

string

DEFAULT

""

GROUP

vhdlio_variables

DESCRIPTION

Use the **vhdllib_logical_name** variable to define the logical name of the VHDL-generated libraries.

If you set the value of this variable to an empty string (""), the base name of the file is used as the default.

This variable is used for VHDL library generation, specifically VITAL models and all VHDL testbenches. Every VHDL model must be analyzed into a logical name that points to a file subdirectory. The .synopsys_vss.setup file defines the mapping between a logical name and the file subdirectory.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdllib_logical_name
```

For a list of all **vhdllib** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

vhdlio_variables(3)

vhdllib_negative_constraint

Determines whether a generated VITAL model is to have negative constraint handling capability.

TYPE

string

DEFAULT

false

GROUP

vhdlio_variables

DESCRIPTION

Determines whether a generated VITAL model is to have negative constraint handling capability.

If you set the value of this variable as true, all cells in the generated VITAL library can handle negative timing constraints. The tool ignores the cell attribute **handle_negative_constraint** value set in the technology library.

If you set the value as false false (the default), only cells in the generated VITAL library whose cell attribute **handle_negative_constraint** is true can handle negative timing constraints. The **vhdllib_negative_constraint** variable does not affect any simulation models other than VITAL.

To see the current value of this variable, type

psyn_shell-xg-t> **printvar vhdllib_negative_constraint**

For a list of all **vhdllib** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

vhdlio_variables(3)

vhdllib_sdf_edge

Determines whether edge information is to be added to delay generics in VITAL libraries.

TYPE

Boolean

DEFAULT

false

GROUP

vhdlio_variables

DESCRIPTION

Use the **vhdllib_sdf_edge** variable to specify whether the tool is to add edge information to delay generics in VITAL libraries. The default value of this variable is false. If you set the value of **vhdllib_sdf_edge** as true, edge information is added.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdllib_sdf_edge
```

For a list of all **vhdllib** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

vhdlio_variables(3)

vhdllib_tb_compare

Controls library testbench generation. No testbenches are created if this variable is set to 0 (the default).

TYPE

Boolean

DEFAULT

0

GROUP

vhdlio_variables

DESCRIPTION

The value you set for the **vhdllib_tb_compare** variable determines whether the tool creates testbenches in your design.

If you set the value of this variable as 0 (the default), no testbenches are created.

If you set the value to a value of from 1 to 5, you are specifying the default value of the testbenches **Cmp_Algorithm** generic integer flag. The greater the number from 1 to 5, the more rigorous the verification.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdllib_tb_compare
```

For a list of all **vhdllib** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

vhdlio_variables(3)

vhdllib_tb_x_eq_dontcare

Specifies the default value for the testbenches **X_Eq_DontCare** generic Boolean flag. If **X_Eq_DontCare** is true, X states are ignored during output comparisons.

TYPE

Boolean

DEFAULT

false

GROUP

vhdlio_variables

DESCRIPTION

Use the **vhdllib_tb_x_eq_dontcare** variable to specify the default value of the testbenches **X_Eq_DontCare** generic Boolean flag. If you set the value of this variable as true, the tool ignores X states during output comparisons. The default value for this variable is false.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdllib_tb_x_eq_dontcare
```

For a list of all **vhdllib** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

vhdlio_variables(3)

vhdllib_timing_checks

Determines the default value of the cell **TimingChecksOn** generic Boolean flag in the VITAL model.

TYPE

Boolean

DEFAULT

true

GROUP

vhdlio_variables

DESCRIPTION

Use the **vhdllib_timing_checks** variable to specify the default value of the cell **TimingChecksOn** generic Boolean flag in the VITAL model. The default value of this variable is true.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdllib_timing_checks
```

For a list of all **vhdllib** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

vhdlio_variables(3)

vhdllib_timing_mesg

Determines the value of the GlitchMode parameter of the **VitalPropagatePathDelay** procedure in the VITAL model.

TYPE

Boolean

DEFAULT

true

GROUP

vhdlio_variables

DESCRIPTION

Use the **vhdllib_timing_mesg** variable to specify the value of the GlitchMode parameter of the **VitalPropagatePathDelay** procedure in the VITAL model. The default value of the **vhdllib_timing_mesg** variable is true.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdllib_timing_mesg
```

For a list of all **vhdllib** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

vhdlio_variables(3)

vhdllib_timing_xgen

Determines the default value of the **XGenerationOn** generic Boolean flag in the VITAL model.

TYPE

Boolean

DEFAULT

false

GROUP

vhdlio_variables

DESCRIPTION

Use the **vhdllib_timing_xgen** variable to specify the default value for the cell **XGenerationOn** generic Boolean flag in the VITAL model. The default value of this variable is false.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdllib_timing_xgen
```

For a list of all **vhdllib** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

vhdlio_variables(3)

vhdllib_vital_99

Determines whether VITAL libraries are to incorporate changes for VITAL 99.

TYPE

Boolean

DEFAULT

false

GROUP

vhdlio_variables

DESCRIPTION

Use the **vhdllib_vital_99** variable to specify whether the VITAL libraries are to incorporate changes for VITAL 99 in your design. The default value of this variable is false.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdllib_vhdllib_vital_99
```

For a list of all **vhdllib** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

vhdlio_variables(3)

vhdlout_bit_type

Sets the basic bit type in a design written to VHDL.

TYPE

string

DEFAULT

std_logic

GROUP

vhdlio_variables

DESCRIPTION

The **vhdlout_bit_type** variable sets the basic bit type in a design written to VHDL. This is useful when you base your design methodology on a logic value system other than std_logic. The default value is std_logic.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdlout_bit_type
```

For a list of all **vhdlout** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

vhdlout_bit_vector_type(3)
vhdlout_one_name(3)
vhdlout_three_state_name(3)
vhdlout_zero_name(3)

vhdlout_bit_vector_type

Sets the basic bit vector type in a design written to VHDL.

TYPE

string

DEFAULT

std_logic_vector

GROUP

vhdlio_variables

DESCRIPTION

The **vhdlout_bit_vector_type** variable sets the basic bit vector type in a design written to VHDL. This is useful when your design methodology is based on a logic value system other than std_logic. The default value is std_logic_vector.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdlout_bit_vector_type
```

For a list of all **vhdlout** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

vhdlout_bit_type(3)
vhdlout_one_name(3)
vhdlout_three_state_name(3)
vhdlout_zero_name(3)

vhdlout_dont_create_dummy_nets

Instructs the VHDL writer not to create dummy nets to connect unused pins or ports in your design.

TYPE

Boolean

DEFAULT

false

GROUP

vhdlio_variables

DESCRIPTION

Setting the value of the **vhdlout_dont_create_dummy_nets** variable as **true** specifies to the VHDL writer that it is not to create dummy nets to connect unused pins or ports in your design,

If you want the VHDL writer to create dummy nets to connect unused pins or ports in your design, set the value of this variable as **false** (the default value).

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdlout_dont_create_dummy_nets
```

For a list of all **vhdlout** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

vhdlio_variables(3)

vhdlout_equations

Defines how the tool is to write combinational logic and sequential logic.

TYPE

Boolean

DEFAULT

false

GROUP

vhdlio_variables

DESCRIPTION

Specifies to the tool how it is to write combinational logic and sequential logic. When you set the value of this variable as **true**, the compiler

- Writes combinational logic as technology-independent Boolean equations, and
- Writes sequential logic as technology-independent wait statements.

If you set the value of this variable as **false** (the default value), the tool writes *all* logic as technology-specific netlists.

Note: If you define a bit type, make sure that Boolean equations are defined for the bit type.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdlout_equations
```

For a list of all **vhdlout** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

vhdlio_variables(3)

vhdlout_follow_vector_direction

Specifies how the tool is to use the original range direction when it writes out an array.

TYPE

Boolean

DEFAULT

true

GROUP

vhdlio_variables

DESCRIPTION

The **vhdlout_follow_vector_direction** variable defines for the tool the way in which the tool is to use the original range direction when it writes out an array.

To achieve this result,

- Set the value of this variable as **true** (the default value), and

- Set the value of the **vhdlout_single_bit** variable to the type mode of **VECTOR**, not the ascending range.

For example, when you set the value of the **vhdlout_follow_vector_direction** variable as **true** (the default value), the VHDL Compiler writes out the correct array range direction, such as 10 down to 0, if the original is 10 down to 0.

Note: If **vhdlout_follow_vector_direction** = **true**, the tool automatically sets the type mode to **VECTOR**.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdlout_follow_vector_direction
```

For a list of all **vhdlout** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

vhdlout_bit_type(3)
vhdlout_bit_vector_type(3)
vhdlout_preserve_hierarchical_types(3)
vhdlout_single_bit(3)

vhdlout_local_attributes

```
This variable is obsolete.
```

SYNTAX

GROUP

DESCRIPTION

vhdlout_lower_design_vector

Determines the way in which the **write -f vhdl** command writes out ports on lower-level designs

TYPE

Boolean

DEFAULT

true

GROUP

vhdlio_variables

DESCRIPTION

The value you specify for the **vhdlout_lower_design_vector** variable determines the way in which the **write -f vhdl** command writes out ports on lower-level designs. A lower-level design is instantiated by any of the designs being written out. The **vhdlout_top_design_vector** variable controls ports on top-level designs.

If you set the value of this variable as **false**, all ports on lower-level designs are written with their original data types. The value of **false** affects only designs read in VHDL format.

If you set the value of this variable as **true** (the default) all ports on lower-level designs are written as bused ports, which means the ports keep their names and are not bit-blasted. The ports are written with the type you define using the **vhdlout_bit_vector_type** variable or, for single-bit ports, using the **vhdlout_bit_type** variable. This setting often results in the most efficient description for simulation. Bus ranges are always in ascending order beginning with 0, regardless of the range description in the original VHDL. If you set this variable as **true**, you must ensure that **vhdlout_bit_vector_type** is defined as an array type whose elements are of **vhdlout_bit_type**.

Note 1 : You cannot set the value of the **vhdlout_lower_design_vector** variable as **false** if the value of the **vhdlout_top_design_vector** variable is set as **true**.

Note 2: If **vhdlout_follow_vector_direction** = true, the value of the **vhdlout_lower_design_vector** variable is automatically set to **true**.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdlout_lower_design_vector
```

For a list of all **vhdlout** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

```
write(2)
vhdlio_variables(3)
vhdlout_bit_type(3)
vhdlout_bit_vector_type(3)
vhdlout_follow_vector_direction(3)
vhdlout_top_design_vector(3)
```

vhdlout_one_name

Determines the literal name for constant bit value 1 in a design written in VHDL.

TYPE

string

DEFAULT

'1'

GROUP

vhdlio_variables

DESCRIPTION

The value you specify for the **vhdlout_one_name** variable determines the literal name for the constant bit value 1 in a design written in VHDL.

This variable is useful when your design methodology is based on a more general logic value than BIT. Use this variable with the **vhdlout_bit_type** variable. The default value is **1**.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdlout_one_name
```

For a list of all **vhdlout** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

vhdlout_bit_type(3)
vhdlout_bit_vector_type(3)
vhdlout_three_state_name(3)
vhdlout_zero_name(3)

vhdlout_package_naming_style

Determines the name the tool is use for the type conversion packages written out by the VHDL writer (VHDLout).

TYPE

string

DEFAULT

CONV_PACK_%d

GROUP

vhdlio_variables

DESCRIPTION

The value you specify for the **vhdlout_package_naming_style** variable determines the name the tool is to use for the type conversion packages written out by the VHDL writer (VHDLout). The value for this variable can contain a string made up of letters, digits, and underscores. The string **%d** is replaced by the name of the current design (**%d** and **%D** are synonymous), as shown as the default value. The tool gives unique names to all conversion packages that are written out.

The tool verifies that the generated name is a valid VHDL identifier and that it is a unique name. If the name is invalid, an error occurs.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdlout_package_naming_style
```

For a list of all **vhdlout** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

vhdlio_variables(3)

vhdlout_preserve_hierarchical_types

Affects the way in which the **write -f vhdl** command writes out ports on lower-level designs

TYPE

string

DEFAULT

VECTOR

GROUP

vhdlio_variables

DESCRIPTION

The value you specify for the **vhdlout_preserve_hierarchical_types** variable affects the way in which the **write -f vhdl** command writes out ports on lower-level designs. A lower-level design is instantiated by any of the designs being written out. Another variable, **vhdlout_single_bit**, controls ports on top-level designs.

Valid values are as follows:

USER

All ports on lower-level designs are written with their original data types. The value of **USER** affects only designs that are read in VHDL format.

VECTOR

All ports on lower-level designs are written with their ports bused, which means that ports keep their names and are not bit-blasted. The ports are written with type defined by **vhdlout_bit_vector_type** or, for single-bit ports, by **vhdlout_bit_type**. This setting of **VECTOR** (the default value) often results in the most efficient description for simulation. Bus ranges are always in ascending order, beginning with 0, regardless of the range description in the original VHDL. If you use **VECTOR**, you must ensure that the value of the **vhdlout_bit_vector_type** variable is an array type whose elements are defined by the **vhdlout_bit_type** variable.

BIT

All typed ports are bit-blasted, which means that a port that is *n* bits wide is written to the VHDL file as *n* separate ports. Each port is given a type as defined by the **vhdlout_bit_type** variable.

Note 1: If **vhdlout_single_bit** = BIT, the tool ignores **vhdlout_preserve_hierarchical_types** and writes out the entire design hierarchy as bit-blasted. Also, **vhdlout_preserve_hierarchical_types** cannot take on a higher value than the current value defined for the **vhdlout_single_bit** variable. The descending order is USER, VECTOR, BIT. Thus, the combination **vhdlout_single_bit** = VECTOR and

vhdlout_preserve_hierarchical_types = USER is not possible.

Note 2: If **vhdlout_follow_vector_direction** = true, the type mode is automatically set to VECTOR.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdlout_preserve_hierarchical_types
```

For a list of all **vhdlout** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

write(2)
vhdlio_variables(3)
vhdlout_bit_type(3)
vhdlout_bit_vector_type(3)
vhdlout_follow_vector_direction(3)
vhdlout_single_bit(3)

vhdlout_separate_scan_in

Affects the way in which the **write -f vhdl** command writes out the scan chain in VHDL.

TYPE

string

DEFAULT

false

GROUP

vhdlio_variables

DESCRIPTION

The value you specify for the **vhdlout_separate_scan_in** variable affects the way in which the **write -f vhdl** command writes out the scan chain in VHDL.

If you set the value of this variable as **false** (the default), the tool writes out the scan chain in the same file as the design. The scan chain is not visible in the testbench, and, therefore, parallel loading is not possible.

If you set the value of this variable as **true**, the tool writes out the scan chain as a package to a separate file. The design must be written out in a single hierarchical file, and it must contain the scan-chain package, before simulation or synthesis takes place.

You receive a message *only* when the scan-chain package is written successfully. Otherwise, no message appears.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdlout_separate_scan_in
```

For a list of all **vhdlout** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

write(2)
vhdlio_variables(3)

vhdlout_single_bit

Affects the way in which the **write -f vhdl** command writes out ports on the top-level design.

TYPE

string

DEFAULT

USER

GROUP

vhdlio_variables

DESCRIPTION

Affects the way in which the **write -f vhdl** command writes out ports on the top-level design.

Valid values are as follows:

USER (false, the default value)
: All ports on the top-level design are written out with their original data types. The option **USER** affects only designs that are read in VHDL format.

VECTOR
: All ports on the top-level design are written with their ports bused, which means that ports keep their names and are not bit-blasted. The ports are written with type defined by the **vhdlout_bit_vector_type** variable, or for single-bit ports, the **vhdlout_bit_type** variable. Lower-level design ports are controlled by the **vhdlout_preserve_hierarchical_types** variable. (A design is lower-level if it is instantiated by any of the designs being written out.) Bus ranges are always in ascending order, beginning with 0, regardless of the range description in the original VHDL. If you use **VECTOR**, you must ensure that the value of the **vhdlout_bit_vector_type** variable is an array type whose elements are defined by the **vhdlout_bit_type** variable.

BIT (true)
: All typed ports are bit-blasted, which means that a port that is *n* bits wide is written to the VHDL file as *n* separate ports. Each port is given a type, as defined by the **vhdlout_bit_type** variable.

Note: If **vhdlout_follow_vector_direction** = true, the type mode is automatically set to VECTOR.

To see the current value of this variable, type

psyn_shell-xg-t> **printvar vhdlout_single_bit**

For a list of all **vhdlout** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

write(2)
vhdlout_bit_type(3)
vhdlout_bit_vector_type(3)
vhdlout_preserve_hierarchical_types(3)

vhdlout_target_simulator

Names the target simulator to which the the tool writes the VHDL file. Valid value is **xp**.

TYPE

string

DEFAULT

""

GROUP

vhdlio_variables

DESCRIPTION

Use the **vhdlout_target_simulator** variable to name the target simulator to which the tool is to write the VHDL file. The only valid value is **xp**. The default is an empty string.

If you set the value of **vhdlout_target_simulator** as **xg**, be sure also to specify the **vhdlout_use_packages** variable to include the following instructions (or the written description will not analyze correctly):

```
SYNOPSYS.attributes.all
```

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdlout_target_simulator
```

For a list of all **vhdlout** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

vhdlout_use_packages(3)
vhdlio_variables(3)

vhdlout_three_state_name

Names the high-impedance bit value used for three-state device values.

TYPE

string

DEFAULT

'Z'

GROUP

vhdlio_variables

DESCRIPTION

Names the high-impedance bit value used for three-state device values. The default value is **Z**.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdlout_three_state_name
```

For a list of all **vhdlout** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

vhdlout_bit_type(3)
vhdlout_bit_vector_type(3)
vhdlout_one_name(3)
vhdlout_zero_name(3)

vhdlout_three_state_res_func

Names a user-supplied three-state resolution function that must be in one of the packages specified by the **vhdlout_use_packages** variable.

TYPE

string

DEFAULT

""

GROUP

vhdlio_variables

DESCRIPTION

The **vhdlout_three_state_res_func** variable names a user-supplied three-state resolution function that must be in one of the packages specified by the **vhdlout_use_packages** variable.

If you set the value of **vhdlout_three_state_res_func** as an empty string ("", the default value), the tool writes out, if needed, a default three-state resolution function. The default three-state resolution function drives a signal to "unknown," if the signal is driven more than once by logic 0 or logic 1.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdlout_three_state_res_func
```

For a list of all **vhdlout** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

write(2)
vhdlout_use_packages(3)
vhdlout_wired_and_res_func(3)
vhdlout_wired_or_res_func(3)

vhdlout_top_configuration_arch_name

Determines the name of the outside architecture, depending on the value you defined for the **vhdlout_write_top_configuration** variable, and causes the VHDL writer (VHDLout) to write out a configuration statement.

TYPE

string

DEFAULT

A

GROUP

vhdlio_variables

DESCRIPTION

The value you define for the **vhdlout_top_configuration_arch_name** variable determines the name of the outside architecture, depending on the value you defined for the **vhdlout_write_top_configuration** variable, and causes the VHDL writer (VHDLout) to write out a configuration statement.

For example, assume the top-level design being written has an entity named Ten and architecture named Tar. The configuration statement binds Ten, Tar, and an outside entity named Oen with an outside architecture named Oar to instantiate Ten as a component. You set the variable as shown in the following example:

```
vhdlout_top_configuration_arch_name = Oar
```

An example of an outside architecture is a testbench the user provides to drive simulation with test vectors.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdlout_top_configuration_arch_name
```

For a list of all **vhdlout** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

write(2)
vhdlout_top_configuration_entity_name(3)
vhdlout_top_configuration_name(3)
vhdlout_write_top_configuration(3)

vhdlout_top_configuration_entity_name

Determines the name of the outside entity, depending on the value you defined for the **vhdlout_write_top_configuration** variable, and causes the VHDL writer (VHDLout) to write out a configuration statement.

TYPE

string

DEFAULT

E

GROUP

vhdlio_variables

DESCRIPTION

The value you specify for the **vhdlout_top_configuration_entity_name** variable determines the name of the outside entity, depending on the value you defined for the **vhdlout_write_top_configuration** variable, and causes the VHDL writer (VHDLout) to write out a configuration statement.

For example, assume the top-level design being written has an entity named Ten and an architecture named Tar. The configuration statement binds Ten, Tar, and an outside entity named Oen with an architecture named Oar, to instantiate Ten as a component. You set the variable as shown in the following example:

```
set vhdlout_top_configuration_entity_name = Oen
```

An example of an outside architecture is a testbench the user provides to drive simulation with test vectors.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdlout_top_configuration_entity_name
```

For a list of all **vhdlout** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

write(2)
vhdlout_top_configuration_arch_name(3)
vhdlout_top_configuration_name(3)
vhdlout_write_top_configuration(3)

vhdlout_top_configuration_name

Determines the name of the configuration statement the **write -f vhdl** command writes out, when the **vhdlout_write_top_configuration** variable is set to **true**.

TYPE

string

DEFAULT

CFG_TB_E

GROUP

vhdlio_variables

DESCRIPTION

The value you specify for the **vhdlout_top_configuration_name** variable determines the name of the configuration statement the **write -f vhdl** command writes out, when the **vhdlout_write_top_configuration** variable is set to **true**.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdlout_top_configuration_name
```

For a list of all **vhdlout** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

write(2)
vhdlout_top_configuration_entity_name(3)
vhdlout_write_top_configuration(3)

vhdlout_top_design_vector

Affects the way in which the **write -f vhdl** command writes out ports on the top-level design.

TYPE

Boolean

DEFAULT

false

GROUP

vhdlio_variables

DESCRIPTION

The value you specify for the **vhdlout_top_design_vector** variable determines the way in which the **write -f vhdl** command writes out ports on top-level designs.

If you set the value of this variable as **false** (the default), all ports on top-level designs are written out with their original data types.

If you set the value of this variable as **true**, all ports on top-level designs are written as bused ports, which means the ports keep their names and are not bit-blasted. The ports are written with the type you define using the **vhdlout_bit_vector_type** variable or, for single-bit ports, using the **vhdlout_bit_type** variable.

Bus ranges are always in ascending order beginning with 0, regardless of the range description in the original VHDL. If you set this variable to **true**, you must ensure that **vhdlout_bit_vector_type** is an array type whose elements are of **vhdlout_bit_type**.

Note: If **vhdlout_follow_vector_direction** = true, the value of the **vhdlout_top_design_vector** variable is automatically set to **true**.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdlout_top_design_vector
```

For a list of all **vhdlout** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

write(2)
vhdlio_variables(3)
vhdlout_bit_type(3)

```
vhdlout_bit_vector_type(3)
vhdlout_follow_vector_direction(3)
vhdlout_lower_design_vector(3)
```

vhdlout_unconnected_pin_prefix

Affects the way in which the **write -f vhdl** command writes out unconnected pin names.

TYPE

string

DEFAULT

n

GROUP

vhdlio_variables

DESCRIPTION

The value you specify for the **vhdlout_unconnected_pin_prefix** variable affects the way in which the **write -f vhdl** command writes out unconnected pin names. You can set the value to any string, but to work with the **change_names** command's default naming prefix, the recommended value for this variable is SYNOPSYS_UNCONNECTED_.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdlout_unconnected_pin_prefix
```

For a list of all **vhdlout** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

write(2)
vhdlout_dont_create_dummy_nets(3)

vhdlout_unknown_name

Specifies the value the tool is to use to drive a signal to the "unknown" state.

TYPE

string

DEFAULT

’X’

GROUP

vhdlio_variables

DESCRIPTION

Use the **vhdlout_unknown_name** variable to specify the value the tool is to use to drive a signal to the "unknown" state. The default value for this variable is **X**.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdlout_unknown_name
```

For a list of all **vhdlout** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

vhdlout_one_name(3)
vhdlout_three_state_name(3)
vhdlout_zero_name(3)

vhdlout_upcase

SYNTAX

```
Obsolete.
```

GROUP

DESCRIPTION

SEE ALSO

vhdlout_use_packages

Instructs the **write -f vhdl** command to write into the VHDL file a clause called a "use clause," which contains a list of package names for each of the packages described in this man page, for all entities.

TYPE

list

DEFAULT

IEEE.std_logic_1164

GROUP

vhdlio_variables

DESCRIPTION

Instructs the **write -f vhdl** command to write into the VHDL file a use clause containing a list of package names, for each of the packages this man page describes, for all entities.

The **write -f vhdl** command also writes out clauses called "library clauses" as needed. If this variable is set to an empty list (""), the variable has no effect on the **write -f vhdl** command, and library clauses are not written out.

The process by which each package is printed out depends on the number of dots (.) in the package name, as follows:

```
  no dots : prepend "work." and append ".all" to the name

  one dot : append ".all" to the name

  two dots: simply print the package name
```

For example:

```
  types     ===> library workuse work.types.all

  my.types===> library myuse my.types.all

  my.types.and===> library myuse my.types.and
```

To see the current value of this variable, type

psyn_shell-xg-t> **printvar vhdlout_use_packages**

For a list of all **vhdlout** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

```
write(2)
vhdlio_variables(3)
```

vhdlout_wired_and_res_func

Specifies the name of a "wired and" resolution function.

TYPE

string

DEFAULT

""

GROUP

vhdlio_variables

DESCRIPTION

Use the **vhdlout_wired_and_res_func** variable to specify the name of a "wired and" resolution function. The name of this user-supplied function must be included in one of the packages the **vhdlout_use_packages** variable specifies. If this variable is set to an empty string, a default "wired and" resolution function is written out, if needed.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdlout_wired_and_res_func
```

For a list of all **vhdlout** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

vhdlio_variables(3)
vhdlout_three_state_res_func(3)
vhdlout_use_packages(3)
vhdlout_wired_or_res_func(3)

vhdlout_wired_or_res_func

Specifies the name of a "wired or" resolution function.

TYPE

string

DEFAULT

""

GROUP

vhdlio_variables

DESCRIPTION

Use the **vhdlout_wired_or_res_func** variable to specify the name of a "wired or" resolution function. The name of this user-supplied function must be included in one of the packages the **vhdlout_use_packages** variable specifies. If this variable is set to an empty string, a default "wired or" resolution function is written out, if needed.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdlout_wired_or_res_func
```

For a list of all **vhdlout** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

vhdlio_variables(3)
vhdlout_three_state_res_func(3)
vhdlout_use_packages(3)
vhdlout_wired_and_res_func(3)

vhdlout_write_architecture

Instructs the **write -format vhdl** command to write out architecture declarations.

TYPE

Boolean

DEFAULT

true

GROUP

vhdlio_variables

DESCRIPTION

The **vhdlout_write_architecture** variable instructs the **write -format vhdl** command to write out architecture declarations. When set to **false**, no architecture declarations are written. The default value is **true**.

Use this variable with the variables **vhdlout_write_entity** and **vhdlout_write_top_configuration** to control the information the **write -format vhdl** command writes out.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdlout_write_architecture
```

For a list of all **vhdlout** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

write(2)
vhdlio_variables(3)
vhdlout_write_entity(3)
vhdlout_write_top_configuration(3)

vhdlout_write_attributes

```
This variable is obsolete.
```

SYNTAX

GROUP

DESCRIPTION

vhdlout_write_components

Instructs the **write -format vhdl** command to write out component declarations for cells mapped to a technology library.

TYPE

Boolean

DEFAULT

true

GROUP

vhdlio_variables

DESCRIPTION

Use the **vhdlout_write_components** variable to instruct the **write -format vhdl** command to write out component declarations for cells mapped to a technology library. The default is **true**. When set to **false**, no component declarations are written. The VHDL format requires component declarations. If you set this variable to **false**, make sure that the **vhdlout_use_packages** variable includes a package containing the necessary component declarations.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdlout_write_components
```

For a list of all **vhdlout** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

write(2)
vhdlio_variables(3)
vhdlout_use_packages(3)

vhdlout_write_entity

Instructs the **write -format vhdl** command to write out entity declarations.

TYPE

Boolean

DEFAULT

true

GROUP

vhdlio_variables

DESCRIPTION

The **vhdlout_write_entity** variable instructs the **write -format vhdl** command to write out entity declarations. The default is **true**. When set to **false**, no entity declarations are written.

Use this variable with the variables **vhdlout_write_architecture** and **vhdlout_write_top_configuration** to control the information the **write -format vhdl** command writes out.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdlout_write_entity
```

For a list of all **vhdlout** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

write(2)
vhdlio_variables(3)
vhdlout_write_architecture(3)
vhdlout_write_top_configuration(3)

vhdlout_write_top_configuration

Instructs the **write -format vhdl** command to write out a configuration statement, if necessary, such as when ports on the top-level design are written as vectors instead of user types.

TYPE

Boolean

DEFAULT

false

GROUP

vhdlio_variables

DESCRIPTION

Use the **vhdlout_write_top_configuration** variable to instruct the **write -format vhdl** command to write out a configuration statement, if necessary, such as when ports on the top-level design are written as vectors instead of user types. The port map of the configuration statement contains calls to type conversion functions. For more information, see the man pages for the variables **vhdlout_single_bit** and **vhdlout_preserve_hierarchical_types**.

If you set the value of this variable as **false** (the default), the tool does not write out a configuration statement.

The configuration statement binds the (written out) top-level design to an entity and architecture outside of the .db file. The outside architecture is assumed to instantiate as a component the written top-level design. An example of an outside architecture is a user-supplied testbench that drives simulation with test vectors.

For example, if the testbench uses user types, but the top-level design's ports are written as vectors, the ports do not interface to the testbench directly because of type mismatches. The use of type conversions in the configuration statement solves this problem. The **write -format vhdl** command does not write a configuration statement for bit-blasted designs.

Inout ports cannot be handled by conversion functions, because there is no way of determining whether to convert for a sink (output) or for a source (input). When **write -format vhdl** encounters an inout port, it checks the netlist. If the port functions as a true bidirectional signal, the tool issues error message VHDL-17. If the inout port functions only as a sink or as a source, **write -format vhdl** creates the correct type conversion.

You can use this variable with the variables **vhdlout_write_architecture** and **vhdlout_write_entity** to control the information **write -format vhdl** writes out.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdlout_write_top_configuration
```

For a list of all **vhdlout** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

write(2)
vhdlout_preserve_hierarchical_types(3)
vhdlout_single_bit(3)
vhdlout_top_configuration_arch_name(3)
vhdlout_top_configuration_entity_name(3)
vhdlout_top_configuration_name(3)
vhdlout_write_architecture(3)
vhdlout_write_entity(3)

vhdlout_zero_name

Determines the literal name for constant bit value 0 in a design written in VHDL.

TYPE

string

DEFAULT

’0’

GROUP

vhdlio_variables

DESCRIPTION

The value you specify for the **vhdlout_zero_name** variable determines the literal name for the constant bit value 0 in a design written in VHDL.

This variable is useful when your design methodology is based on a more general logic value than BIT. Use this variable with the **vhdlout_bit_type** variable. The default value is 0.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar vhdlout_zero_name
```

For a list of all **vhdlout** variables and their current values, see the **vhdlio_variables**(3) man page.

SEE ALSO

write(2)
vhdlout_bit_type(3)
vhdlout_bit_vector_type(3)
vhdlout_one_name(3)
vhdlout_three_state_name(3)

via_attributes

Contains attributes related to via.

DESCRIPTION

Contains attributes related to via.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified object. Specified with **list_attribute -class via -application**, the definition of attributes can be listed.

Via Attributes

array_size

Specifies array size of a via object. Its format is {**col row**}.
The data type of **array_size** is string.
This attribute is read-only.

bbox

Specifies the bounding-box of a via. The **bbox** is represented by a **rectangle**.
The format of a *rectangle* specification is {{llx lly} {urx ury}}, which specifies the lower-left and upper-right corners of the rectangle.
The data type of **bbox** is string.
This attribute is read-only.

bbox_ll

Specifies the lower-left corner of the bounding-box of a via.
The **bbox_ll** is represented by a **point**. The format of a *point* specification is {x y}.
You can get the **bbox_ll** of a via, by accessing the first element of its **bbox**.
The data type of **bbox_ll** is string.
This attribute is read-only.

bbox_llx

Specifies x coordinate of the lower-left corner of the bounding-box of a via.
The data type of **bbox_llx** is integer.
This attribute is read-only.

bbox_lly

Specifies y coordinate of the lower-left corner of the bounding-box of a via.
The data type of **bbox_lly** is integer.
This attribute is read-only.

bbox_ur

Specifies the upper-right corner of the bounding-box of a via.
The **bbox_ur** is represented by a **point**. The format of a *point* specification is {x y}.
You can get the **bbox_ur** of a via, by accessing the second element of its **bbox**.
The data type of **bbox_ur** is string.
This attribute is read-only.

bbox_urx

Specifies x coordinate of the upper-right corner of the bounding-box of a via.
The data type of **bbox_urx** is integer.
This attribute is read-only.

bbox_ury

Specifies y coordinate of the upper-right corner of the bounding-box of a via.
The data type of **bbox_ury** is integer.
This attribute is read-only.

cell_id

Specifies Milkyway design ID in which a via object is located.
The data type of **cell_id** is integer.
This attribute is read-only.

center

Specifies the center position of a via object.
The data type of **center** is string.
This attribute is read-only.

col

Specifies number of horizontal columns of a via object.
The data type of **col** is integer.
This attribute is read-only.

layer

Specifies layer name list with which a via object is associated. Its format is {**via_layer lower_layer upper_layer**}.
The data type of **layer** is string.
This attribute is read-only.

lower_layer

Specifies lower layer name.
The data type of **lower_layer** is string.
This attribute is read-only.

name

Specifies name of a via object.
The data type of **name** is string.
This attribute is read-only.

net_id

Specifies object ID of the net associated with a via object.
The data type of **net_id** is integer.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

net_type

Specifies type of net associated with a via object.
The data type of **net_type** is string.
This attribute is read-only.

object_class

Specifies object class name of a via object, which is **via**.
The data type of **object_class** is string.

This attribute is read-only.

object_id

Specifies object ID in Milkyway design file.
The data type of **object_id** is integer.
This attribute is read-only.

object_type

Specifies object type name, which can be **via**, **via_array** or **via_cell**.
The data type of **object_type** is string.
This attribute is read-only.

orientation

Specifies orientation of a via object.
The data type of **orientation** is string.
Its valid values are:

- **N**
- **E**
- **S**
- **W**
- **FN**
- **FE**
- **FS**
- **FW**

This attribute is writable. You can use **set_attribute** to modify its value on a specified object. However when object type is via_cell, you can't change its **orientation**.

owner

Specifies Milkyway design file name in which a via is located.
The data type of **owner** is string.
This attribute is read-only.

owner_net

Specifies net name which a via is connected to.
The data type of **owner_net** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

route_type

Specifies route type of a via.
The data type of **route_type** is string.
Its valid values are:

- **User Enter** or **user_enter**
- **Signal Route** or **signal_route**
- **Signal Route (Global)** or **signal_route_global**
- **P/G Ring** or **pg_ring**
- **Clk Ring** or **clk_ring**
- **P/G Strap** or **pg_strap**
- **Clk Strap** or **clk_strap**
- **P/G Macro/IO Pin Conn** or **pg_macro_io_pin_conn**
- **P/G Std. Cell Pin Conn** or **pg_std_cell_pin_conn**
- **Zero-Skew Route** or **clk_zero_skew_route**
- **Bus** or **bus**
- **Shield (fix)** or **shield**
- **Shield (dynamic)** or **shield_dynamic**
- **Fill Track** or **clk_fill_track**
- **Unknown** or **unknown**

It is determined by Tcl variable mw_attr_value_no_space whether **get_attribute** or **report_attribute** returns route_type containing spaces or underscores. This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

row
: Specifies number of vertical rows in a via object.
The data type of **row** is integer.
This attribute is read-only.

upper_layer
: Specifies upper layer name.
The data type of **upper_layer** is string.
This attribute is read-only.

via_layer
: Specifies via layer name.
The data type of **via_layer** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

via_master
: Specifies the via master‚Äôs name defined in the library‚Äôs technology file; for via_cell, it is the library cell’s name.
The data type of **via_master** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

x_pitch
: Specifies the center-to-center spacing of a via object in the horizontal direction.
The data type of **x_pitch** is float.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

y_pitch
: Specifies the center-to-center spacing of a via object in the vertical direction.
The data type of **y_pitch** is float.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

SEE ALSO

get_attribute(2),
list_attribute(2),
mw_attr_value_no_space(3),
report_attribute(2),
set_attribute(2).

view_analyze_file_suffix

Specifies, in a list of file extensions, the files shown in the File/Analyze dialog box of Design Analyzer.

TYPE

string

DEFAULT

{v vhd vhdl}

GROUP

suffix_variables

DESCRIPTION

Specifies, in a list of file extensions, the files shown in the File/Analyze dialog box of Design Analyzer. The default value is {*v*, *vhd*, *vhdl*}.

To determine the current value of this variable, type **printvar view_analyze_file_suffix**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_variables (3).

view_arch_types

Sets the contents of the architecture option menu. Contains a list of host machine architectures you can use for background jobs from the Design Analyzer viewer.

TYPE

list

DEFAULT

sparcOS5 hpux10 rs6000 sgimips

GROUP

view_variables

DESCRIPTION

Sets the contents of the architecture option menu. Contains a list of host machine architectures you can use for background jobs from the Design Analyzer viewer.

To determine the current value of this variable, use **printvar view_arch_types**. For a list of all **view** variables and their current values, use the **print_variable_group view** command.

SEE ALSO

view_variables(3)

view_background

Specifies the background color of the Design Analyzer viewer.

TYPE

string

DEFAULT

black

GROUP

view_variables

DESCRIPTION

Specifies the background color of the Design Analyzer viewer. Valid settings are *white* (the default) and *black*.

To determine the current value of this variable, type **printvar view_background**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_variables(3)

view_cache_images

Specifies to Design Analyzer that the tool is to cache bitmaps for fast schematic drawing.

TYPE

string

DEFAULT

true

GROUP

view_variables

DESCRIPTION

Specifies that the tool is to cache bitmaps for fast schematic drawing. The default is *true*.

To determine the current value of this variable, type **printvar view_cache_images**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_variables (3).

view_command_log_file

Names a file and its location that is to contain all text written to the Design Analyzer Command window.

TYPE

string

DEFAULT

"./view_command.log"

GROUP

view_variables

DESCRIPTION

Names a file and its location that is to contain all text written to the Design Analyzer Command window. The default is to set this variable.

To determine the current value of this variable, use **printvar view_command_log_file**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

command_log_file(3)
view_log_file(3)

view_command_win_max_lines

Contains the maximum number of lines to be saved in the Design Analyzer command window.

TYPE

integer

DEFAULT

1000

GROUP

view_variables

DESCRIPTION

Contains the maximum number of lines to be saved in the Design Analyzer command window. When the number of lines of output added to the command window exceed the number this variable specifies, the older lines at the top of the list are removed.

This variable should be set to 1000 or more. Values up to tens of thousands are also useful.

To determine the current value of this variable, type **printvar view_command_win_max_lines**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_variables (3).

view_dialogs_modal

Requires that the question and error dialogs in Design Analyzer be confirmed, before you can continue entering commands.

TYPE

Boolean

DEFAULT

true

GROUP

view_variables

DESCRIPTION

Requires that the question and error dialogs in Design Analyzer be confirmed, before you can continue entering commands.

To determine the current value of this variable, use **printvar view_dialogs_modal**. For a list of all **view** variables and their current values, type the **print_variable_group view**.

SEE ALSO

view_variables (3).

view_disable_cursor_warping

Causes the cursor to be automatically "warped" (moved). When *false*, the cursor is automatically "warped" (or moved) to dialog boxes.

TYPE

Boolean

DEFAULT

true

GROUP

view_variables

DESCRIPTION

Causes the cursor to be automatically "warped" (moved). When *false*, the cursor is automatically "warped" (or moved) to dialog boxes. The default is *true*.

To determine the current value of this variable, use **printvar view_disable_cursor_warping**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_variables (3).

view_disable_error_windows

Instructs Design Analyzer not to post the error windows when errors occur.

TYPE

Boolean

DEFAULT

false

GROUP

view_variables

DESCRIPTION

Instructs Design Analyzer not to post the error windows when errors occur. The default is *false*.

To determine the current value of this variable, type **printvar view_disable_error_windows**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_variables (3).

view_disable_output

Disables output to the Design Analyzer command window.

TYPE

Boolean

DEFAULT

false

GROUP

view_variables

DESCRIPTION

Disables output to the Design Analyzer command window, when set to *true*. This variable is useful when you run Design Analyzer over slow networks, such as telephone lines. The default value is *false*.

To determine the current value of this variable, type **printvar view_disable_output**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_variables (3).

view_error_window_count

Specifies the maximum number of errors Design Analyzer reports for a command.

TYPE

integer

DEFAULT

6

GROUP

view_variables

DESCRIPTION

Specifies the maximum number of errors Design Analyzer reports for a command. The default value is *6*. If more than the specified number of errors occurs, you are informed that you can see additional errors in the command window. The error window is suppressed until the end of the command.

To display all errors, set this variable to 0. To display no errors, set this variable to a negative number or set the **view_disable_error_windows** variable to *true*.

To determine the current value of this variable, type **printvar view_error_window_count**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_disable_error_windows (3), **view_variables** (3).

view_execute_script_suffix

Displays only files with the stated suffixes, from directories you select in the Execute Script option window of the Setup menu of Design Analyzer.

TYPE

list

DEFAULT

".script .scr .dcs .dcv .dc .con .tcl"

GROUP

suffix_variables

DESCRIPTION

Displays only files with the stated suffixes, from directories you select in the Execute Script option window of the Setup menu of Design Analyzer.

To determine the current value of this variable, type **printvar view_execute_script_suffix**. For a list of all **suffix** variables and their current values, type **print_variable_group suffix**.

SEE ALSO

suffix_variables(3)

view_info_search_cmd

Invokes, if set, the online information viewer through the optional menu item On-Line Information.

TYPE

string

DEFAULT

""

GROUP

view_variables

DESCRIPTION

Invokes, if set, the online information viewer through the optional menu item On-Line Information. Set the value of this variable to the UNIX path name to the online information viewer.

To determine the current value of this variable, type **printvar view_info_search_cmd**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_variables (3).

view_log_file

Specifies the file in which the tool stores events that occur in the viewer.

TYPE

string

DEFAULT

""

GROUP

view_variables

DESCRIPTION

Specifies the file in which the tool stores events that occur in the viewer. This variable is useful for error reporting. You can execute this variable with the Execute Script option of the Setup menu, or insert the file, using the **include** command in Design Analyzer.

To determine the current value of this variable, type **printvar view_log_file**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

include (2); **view_variables** (3).

view_on_line_doc_cmd

Invokes, if set, the online documentation viewer, through the optional menu item On-Line Documentation.

TYPE

string

DEFAULT

""

GROUP

view_variables

DESCRIPTION

Invokes, if set, the online documentation viewer, through the optional menu item On-Line Documentation. Set the value of this variable to the UNIX command to invoke the online documentation view: for example, /vobs/snps/synopsys/sold.

To determine the current value of this variable, type **printvar view_on_line_doc_cmd**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_variables (3).

view_read_file_suffix

Displays only files with the stated suffixes, from directories you select with the Read option of the File menu of Design Analyzer.

TYPE

list

DEFAULT

db gdb sdb edif eqn fnc lsi mif NET pla st tdl v vhd vhdl xnf

GROUP

suffix_variables

DESCRIPTION

Displays only files with the stated suffixes, from directories you select with the Read option of the File menu of Design Analyzer.

To determine the current value of this variable, type **printvar view_read_file_suffix**. For a list of all **suffix** variables and their current values, type **print_variable_group suffix**.

SEE ALSO

suffix_variables(3)

view_report_append

Specifies to the tool to append to the specified file the reports the Design Vision menus generate.

TYPE

Boolean

DEFAULT

true

GROUP

view_variables

DESCRIPTION

Specifies to the tool to append to the specified file the reports the Design Vision menus generate. To achieve this result, set the value of this variable to *true* (the default).

For a list of all **view** variables and their current values, type **print_variable_group view**.

To determine the current value of this variable, enter one of the following commands, depending on which mode you are using:

```
      prompt> printvar view_report_append
      or
            dc_shell-t> printvar view_report_append
```

SEE ALSO

view_variables(3)

view_report_interactive

Specifies to the tool to send to the command line view the reports generated by Design Vision menus.

TYPE

Boolean

DEFAULT

true

GROUP

view_variables

DESCRIPTION

Specifies to the tool to send to the command line view the reports generated by Design Vision menus. To achieve this result, set the value of this variable to *true*.

For a list of all **view** variables and their current values, type **print_variable_group view**.

To determine the current value of this variable, type one of the following commands, depending on which mode you are using:

```
      prompt> printvar view_report_interactive
      or
            dc_shell-t> printvar view_report_interactive
```

SEE ALSO

view_variables(3)

view_report_output2file

Specifies to the tool to send to the specified file the reports generated by Design Vision menus.

TYPE

Boolean

DEFAULT

false

GROUP

view_variables

DESCRIPTION

Specifies to the tool to send to the specified file the reports generated by Design Vision menus.

To achieve this result, set the value of this variable to *true*. The default is *false*.

For a list of all **view** variables and their current values, type **print_variable_group view**.

To determine the current value of this variable, type one of the following commands, depending on which mode you are using:

```
      prompt> printvar view_report_output2file
      or
            dc_shell-t> printvar view_report_output2file
```

SEE ALSO

view_variables(3)

view_script_submenu_items

Allows users to add to the Design Analyzer Setup pulldown menu valid items to invoke user scripts.

TYPE

string

DEFAULT

{}

GROUP

view_variables

DESCRIPTION

Allows users to add to the Design Analyzer Setup pulldown menu valid items to invoke user scripts. The variable should contain a list of strings, grouped into pairs. The first member of the pair is the text that will appear in the submenu. The second member is the string that gets sent to the dc_shell command line for execution. You can use any valid dc_shell command sequence.

For example,

```
view_script_submenu_items = "{"List", "list_instances", "ls", "sh ls -lR", "Update",
"include update.dcsh"}"
```

creates a submenu under the Scripts menu item on the Setup pulldown menu. The submenu contains the strings List, ls, and Update. Selecting one of these entries executes the commands **list_instances**, **sh ls -lR**, or **include update.dcsh**, respectively.

The tool reads this variable only at startup time, so any changes after the Design Analyzer is initialized are not reflected.

To determine the current value of this variable, type **list view_script_submenu_items**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_variables (3).

view_set_selecting_color

Specifies the color to use for selecting and zooming.

TYPE

string

GROUP

view_variables

DESCRIPTION

Specifies the color to use for selecting and zooming. You can set this variable in the **.synopsys_dc.setup** initialization file. Any color in **/usr/lib/X11/rgb.txt** is valid.

To determine the current value of this variable, type **printvar view_set_selecting_color**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_variables (3).

view_tools_menu_items

Permits partial configuration of the Tools pulldown menu to add a new menu item for invoking user scripts.

TYPE

string

DEFAULT

{}

GROUP

view_variables

DESCRIPTION

Permits partial configuration of the Tools pulldown menu to add a new menu item for invoking user scripts. This .synopsys_dc.setup file variable contains a list of strings, grouped into pairs. The first member of the pair is the text that appears in the submenu. The second member is the string that is sent to the dc_shell command line for execution. You can use any valid dc_shell command sequence.

For example,

```
view_tools_menu_items = "{"List", "list_instances", "ls", "sh ls -lR", "Update",
"include update.dcsh"}"
```

adds three more items to the Tools menu: List, ls, and Update. Selecting one of these entries executes one of the commands. For instance, if you selected **update**, the **include update.dcsh** command would be executed.

The tool reads this variable only at start-up time. Any changes after the Design Analyzer is initialized are not reflected.

To determine the current value of this variable, type **printvar view_tools_menu_items**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_variables (3).

view_use_small_cursor

Specifies to the tool that the X display is to support only 16 x 16-bit map size cursors.

TYPE

string

DEFAULT

""

GROUP

view_variables

DESCRIPTION

Specifies to the tool that the X display is to support only 16 x 16-bit map size cursors. To achieve this result, set the value of this variable to *true*. The default is "".

To determine the current value of this variable, type **printvar view_use_small_cursor**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_variables (3).

view_use_x_routines

Enables the use of internal arc-drawing routines (instead of X routines).

TYPE

Boolean

DEFAULT

true

GROUP

view_variables

DESCRIPTION

Enables the use of internal arc-drawing routines (instead of X routines). If there is a math coprocessor chip on the same machine that the X server is on, X arc-drawing routines are faster. Otherwise, internal arc-drawing routines are faster. The default value is *true*.

To determine the current value of this variable, type **printvar view_use_x_routines**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_variables (3).

view_variables

Directly affect the graphic display used by the Design Analyzer viewer.

SYNTAX

```
string default_schematic_options = "-size infinite"
string test_design_analyzer_uses_insert_scan = "true"
string text_editor_command = "xterm -fn 8x13 -e vi %s &"
string text_print_command = "lpr -Plw"
string view_analyze_file_suffix =  {"v", "vhd", "vhdl"}
list view_arch_types = {"apollo", "decmips", "hp700", "mips", "necmips", "rs6000",
                             "sgimips", "sonymips", "sparc"}
string view_background = "black"
string view_cache_images = "true"
string view_command_log_file = "./view_command.log"
integer view_command_win_max_lines = 1000
Boolean view_dialogs_modal = true
Boolean view_disable_cursor_warping = "true"
Boolean view_disable_error_windows = "false"
Boolean view_disable_output = "false"
integer view_error_window_count = 6
list view_execute_script_suffix = {".script", ".scr", ".dcs", ".dcsh", ".dc",
                                        ".con", ".wscr", ".rscr"}
string  view_info_search_cmd = "/remote/src/syn/ice/cbu.slot3/infosearch/scripts/
InfoSearch"
string view_log_file = ""
string view_on_line_doc_cmd = ""
list view_read_file_suffix = {"db", "sdb", "edif", "eqn", "fnc", "lsi", "mif",
                                   "NET", "pla", "st", "tdl", "v", "vhd", "vhdl"}
string view_script_submenu_items = {"DA to SGE Transfer", "write_sge"}
list  view_tools_menu_items =   {}
string view_use_small_cursor = ""
string view_use_x_routines = "true"
list view_write_file_suffix = {"db", "sdb", "do", "edif", "eqn", "fnc", "lsi",
                              "NET", "neted", "pla", "st", "tdl", "v", "vhd", "vhdl
", "xnf"}
string x11_set_cursor_background = ""
string x11_set_cursor_foreground = ""
integer x11_set_cursor_number = "-1"

Boolean view_report_interactive = "true"
Boolean view_report_output2file = "false"
Boolean view_report_append = "false"
```

DESCRIPTION

Directly affect the graphic display used by the Design Analyzer viewer. Defaults are shown above, under "Syntax."

For a list of **view** variables and their current values, type **print_variable_group view**. To view this manual page online, type **help view_variables**. To view an individual variable description, type **help *var***, where *var* is the name of the variable.

default_schematic_options
: Specifies options to use when schematics are generated. When set to "-size infinite" (default), the schematic for a design is displayed on a single page. Used by the Design Analyzer.

test_design_analyzer_uses_insert_scan
: When *true* (the default), **design_analyzer** executes **insert_scan** when the Tools/Test Synthesis.../Insert Internal Scan Circuitry... menu is selected in the Design Analyzer text window and the OK button is pushed. When *false*, **design_analyzer** executes **insert_dft** instead of **insert_scan**.

text_editor_command
: Specifies the command that executes when the **Edit/File** menu is selected in the Design Analyzer text window.

text_print_command
: Specifies the command that executes when the **File/Print** menu is selected in the Design Analyzer text window.

view_analyze_file_suffix
: This variable is a list of file extensions that specifies the files that are shown in the File/Analyze dialog. Its default value is "{.vhd, .v, .vhdl}".

view_arch_types
: List of host machine architectures that can be used for background jobs from the Design Analyzer viewer. This variable is used to set the contents of the architecture option menu.

view_background
: Specifies the background color of the Design Analyzer viewer. The valid settings are "black" (default) and "white".

view_cache_images
: Specifies whether bit maps are to be cached for fast schematic drawing. Default value is *true*.

view_command_log_file
: When set, all text written to the Design Analyzer Command Window is written to the specified file.

view_command_win_max_lines
: The maximum number of lines to be saved in the Design Analyzer command window. When a larger number of lines of output are added to the command window, the older lines at the top of the list are removed.

view_dialogs_modal
: When *true*, the question and error dialogs in Design Analyzer require a confirmation before you can continue to enter commands. The default is *true*.

view_disable_cursor_warping
: When *false*, the cursor is automatically "warped" (or moved) to dialogs when they are posted. Default is *true*.

view_disable_error_windows
When *true*, error windows are not posted when errors occur. Default is *false*.

view_disable_output
This variable specifies whether output to the Design Analyzer Command Window is to be disabled. This is useful when running the Design Analyzer over slow networks, like telephone lines. The default value is *false*.

view_error_window_count
Maximum number of errors that Design Analyzer reports for a command. If more than the specified number of errors occurs, you are informed that additional errors can be seen in the command window. The error window is suppressed until the end of the command. Default is 6.

view_execute_script_suffix
Used by the "Execute Script" option of the Setup menu, it displays only files with these suffixes from directories selected in the option window.

view_info_search_cmd
If this variable is set, the optional menu item "On-Line Information" allows you to invoke the online information viewer. The value of this variable should be set to the UNIX pathname to the online information viewer.

view_log_file
Specifies the file where events that occur in the viewer are stored. This file is useful for error reporting and can be executed with the "Execute Script" option of the Setup menu, or inserted with the **include** command in the Design Analyzer.

view_on_line_doc_cmd
If this variable is set, the optional menu item, called "On-Line Documentation," allows you to invoke the on-line documentation viewer. The value of this variable should be set to the UNIX command to invoke the on-line documentation view.

view_read_file_suffix
Used by the "Read" option of the Design Analyzer File menu. Displays only files with these suffixes from directories that you select in the option window.

view_script_submenu_items
A **.synopsys_dc.setup** file variable that permits partial configuration of the Setup pulldown menu to add a new menu item for invoking user scripts.

view_tools_menu_items

A **.synopsys_dc.setup** file variable that permits partial configuration of the Tools pulldown menu to add a new menu item for invoking user scripts. Contains a list of strings, grouped into pairs. The first member of the pair is the text that will appear in the submenu. The second member is the string that is sent to the **dc_shell** command line for execution.

view_use_small_cursor
Is *true* if the X-display supports only 16-bit (small) cursors.

view_use_x_routines

When *false*, internal arc-drawing routines (instead of X routines) are used. If there is a math co-processor chip on the same machine that the X server is on, X arc-drawing routines are faster. Otherwise, internal arc-drawing routines are faster. Default is *true*.

view_write_file_suffix

Used by the "Save As" option of the File menu. Displays only files with these suffixes from directories selected in the option window.

x11_set_cursor_background

Specifies background color of the cursor in the Design Analyzer menus and viewer. This variable can be set in the **.synopsys_dc.setup** initialization file. Any color in **/usr/lib/X11/rgb.txt** is valid.

x11_set_cursor_foreground

Specifies foreground color of the cursor in the Design Analyzer menus and viewer. This variable can be set in the **.synopsys_dc.setup** initialization file. Any color in **/usr/lib/X11/rgb.txt** is valid.

x11_set_cursor_number

Specifies the cursor from the standard X cursor font used by the Design Analyzer menus and viewer. If this variable is not set, or is set to "-1", the cursor used by the X background is also used for all windows and menus. This variable can be set in the **.synopsys_dc.setup** initialization file.

view_report_interavtive

When set to *true*, reports generated via Design Vision’s menus are sent to the command line view.

view_report_output2file

When set to *true*, reports generated via Design Vision’s menus are sent to the file specified via the report dialog box.

view_report_append

view_report_append: When set to *true*, reports generated via Design Vision’s menus are appended to the file specified via the report dialog box.

SEE ALSO

design_analyzer (1); **create_schematic** (2).

view_write_file_suffix

Displays only files with the stated suffixes, from directories you select with the Save As option of the File menu of Design Analyzer.

TYPE

list

DEFAULT

gdb db sdb do edif eqn fnc lsi NET neted pla st tdl v vhd vhdl xnf

GROUP

suffix_variables

DESCRIPTION

Displays only files with the stated suffixes, from directories you select with the Save As option of the File menu of Design Analyzer.

To determine the current value of this variable, type **printvar view_write_file_suffix**. For a list of all **suffix** variables and their current values, type **print_variable_group suffix**.

SEE ALSO

suffix_variables(3)

voltage_area_attributes

Contains attributes related to voltage area.

DESCRIPTION

Contains attributes related to voltage area.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified object. Specified with **list_attribute -class voltage_area -application**, the definition of attributes can be listed.

Voltage Area Attributes

bbox

Specifies the bounding-box of a voltage area. The **bbox** is represented by a **rectangle**.
The format of a *rectangle* specification is {{llx lly} {urx ury}}, which specifies the lower-left and upper-right corners of the rectangle.
The data type of **bbox** is string.
This attribute is read-only.

bbox_ll

Specifies the lower-left corner of the bounding-box of a voltage area.
The **bbox_ll** is represented by a **point**. The format of a *point* specification is {x y}.
You can get the **attr_name** of a voltage area, by accessing the first element of its **bbox**.
The data type of **bbox_ll** is string.
This attribute is read-only.

bbox_llx

Specifies x coordinate of the lower-left corner of the bounding-box of a voltage area.
The data type of **bbox_llx** is double.
This attribute is read-only.

bbox_lly

Specifies y coordinate of the lower-left corner of the bounding-box of a voltage area.
The data type of **bbox_lly** is double.
This attribute is read-only.

bbox_ur

Specifies the upper-right corner of the bounding-box of a voltage area.
The fbbox_ur is represented by a **point**. The format of a *point* specification is {x y}.
You can get the **bbox_ur** of a voltage area, by accessing the second element of its **bbox**.
The data type of **bbox_ur** is string.
This attribute is read-only.

bbox_urx

Specifies x coordinate of the upper-right corner of the bounding-box of a voltage area.
The data type of **bbox_urx** is double.
This attribute is read-only.

bbox_ury

Specifies y coordinate of the upper-right corner of the bounding-box of a voltage area.
The data type of **bbox_ury** is double.
This attribute is read-only.

cell_id

Specifies Milkyway design ID in which a voltage area object is located.
The data type of **cell_id** is integer.
This attribute is read-only.

color

Specifies the color for a voltage area and its leaf cells.
The data type of **color** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

guardband

Specifies the guardband of a voltage area.
Its format is {**guardband_x guardband_y**}.
Guardband is the spacing along the boundary of a voltage area where cells cannot be placed because of the lack of power supply rails.
The data type of **guardband** is string.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

guardband_x

Specifies the guardband width in the horizontal direction.
The data type of **guardband_x** is integer.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

guardband_y

Specifies the guardband width in the vertical direction.
The data type of **guardband_y** is integer.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

is_fixed

Specifies whether a voltage area is in a fixed location, and the shaping will ignore it.
The data type of **is_fixed** is boolean.
This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

modules

Specifies collection of the top node cells inside a voltage area.
The data type of **modules** is collection.
This attribute is read-only.

name

   Specifies name of a voltage area object.
   The data type of **name** is string.
   This attribute is read-only.

object_class

   Specifies object class name of a voltage area, which is **voltage_area**.
   The data type of **object_class** is string.
   This attribute is read-only.

object_id

   Specifies object ID in Milkyway design file.
   The data type of **object_id** is integer.
   This attribute is read-only.

points

   Specifies point list of a voltage area's boundary.
   The data type of **points** is string.
   This attribute is read-only.

utilization

   Specifies utilization of a voltage area.
   The data type of **utilization** is float.
   This attribute is read-only.

SEE ALSO

get_attribute(2),
list_attribute(2),
report_attribute(2),
set_attribute(2).

wildcards

Describes supported wildcard characters and ways to escape them.

DESCRIPTION

Design Compiler supports following characters as wildcards.

Asterisks (*) substitute for a string of characters of any length.
Question marks (?) substitute for a single character.

Following commands support wildcards.

```
find
get_cells
get_clocks
get_designs
get_lib_cells
get_lib_pins
get_libs
get_multibits
get_nets
get_pins
get_ports
get_references
list_designs
list_instances
list_libs
trace_nets
untrace_nets
```

In addition to these, commands that perform an implicit find also support wildcarding feature.

Escaping Wildcards

Wildcard characters must be escaped using double backslashes (\\) to remove their special regular expression meaning. Refer to EXAMPLES section of this manual page for more information.

Escaping Escape Character (\\)

This is similar to that of escaping wildcard characters, but needs one escape character each to escape the escape character. Refer to EXAMPLES section of this manual page for more information.

EXAMPLES

Using Wildcards

The following example finds all nets in the current design which are prefixed by in and followed by any two characters.

```
dc_shell> find(net, in??)
{"in11", "in21"}
```

The equivalent command in Tcl-based dc_shell is:

```
dc_shell-t> get_nets in??
{"in11", "in21"}
```

The following example finds all cells in the current design which are prefixed by U and followed by a string of characters of any length.

```
dc_shell> find(cell, "U*")
{"U1", "U2", "U3", "U4"}
```

The equivalent command in Tcl-based dc_shell is:

```
dc_shell-t> get_cells U*
{"U1", "U2", "U3", "U4"}
```

Escaping Wildcards

The following example finds design test?1 in the system.

```
dc_shell> find(design, "test\\?1")
{"test?1"}
```

The equivalent command in Tcl-based dc_shell is:

```
dc_shell-t> get_designs {test\\?1}
{"test?1"}
```

The same above example can be used in Tcl-based dc_shell using the Tcl **list** command.

```
        dc_shell-t> get_designs [list {test\?1}]
        {"test?1"}
```

If neither curly braces nor **list** is used in Tcl-based dc_shell, then the syntax is as below:

```
        dc_shell-t> get_designs test\\\\?1
        {"test?1"}
```

Escaping Escape Character (\\)

The following example finds design test\1 in the system.

```
        dc_shell> find(design, "test\\\\1")
        {"test\1"}
```

The equivalent command in Tcl-based dc_shell is:

```
        dc_shell-t> get_designs {test\\\\1}
        {"test\1"}
```

The same above example can be used in Tcl-based dc_shell using the Tcl **list** command.

```
        dc_shell-t> get_designs [list {test\\1}]
        {"test\1"}
```

If neither curly braces nor **list** is used in Tcl-based dc_shell, then the syntax is as below:

```
        dc_shell-t> get_designs test\\\\\\\\1
        {"test\1"}
```

write_name_nets_same_as_ports

Specifies to the tool that nets are to receive the same names as the ports the nets are connected to.

TYPE

Boolean

DEFAULT

false

GROUP

edif_variables, io_variables

DESCRIPTION

Specifies to the tool that nets are to receive the same names as the ports the nets are connected to. To achieve this result, set the value of this variable to *true*. This variable affects nets in design descriptions written in EDIF, LSI, or TDL format. (Other nets might be renamed to avoid creating shorts.) The default value of this variable is *false*.

Note: In the EDIF format, even if the **edifout_power_and_ground_representation** variable is set to *port*, the power and ground nets will not be named the same as the power and ground ports that are written in the interface construct of each cell construct.

Net names in the design database are unchanged; that is, only the file being written out is affected and not the original design data.

To determine the current value of this variable, type **printvar write_name_nets_same_as_ports**. For a list of all **edif** or **io** variables and their current values, type **print_variable_group edif** or **print_variable_group io**.

SEE ALSO

edif_variables(3)
io_variables(3)

write_sdc_output_lumped_net_capacitance

Determines whether or not the **write_sdc** command outputs net loads.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When true, the write_sdc command outputs net loads. When false, write_sdc does not output net loads. The net loads are output thru set_load statements during write_sdc.

The default value of the variable is true. That is, by default all net loads will be output during write_sdc execution.

SEE ALSO

set_load(2)
write_sdc(2)

write_sdc_output_net_resistance

Determines whether or not the **write_sdc** command outputs net resistance.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When true, the write_sdc command outputs net resistance. When false, write_sdc does not output net resistance. The net resistance are output thru set_resistance statements during write_sdc.

The default value of the variable is true. That is, by default all net resistance will be output during write_sdc execution.

SEE ALSO

set_resistance(2)
write_sdc(2)

write_test_formats

Specifies the test vector formats recognized and created by the **write_test** command.

TYPE

list

DEFAULT

synopsys tssi_ascii tds verilog vhdl wgl

GROUP

write_test_variables

DESCRIPTION

Specifies the test vector formats recognized and created by the **write_test** command.

To determine the current value of this variable, type **printvar write_test_formats**. For a list of all **write_test** variables and their current values, type **print_variable_group write_test**.

SEE ALSO

write_test(2)
write_test_variables(3)

write_test_include_scan_cell_info

Specifies to the **write_test** command to include in the vector files scan-chain, cell, and inversion information for vector formats.

TYPE

string

DEFAULT

true

GROUP

write_test_variables

DESCRIPTION

Specifies to the **write_test** command to include in the vector files scan-chain, cell, and inversion information for vector formats. To achieve this result, set this variable to *true* (the default).

For a simulator to execute a parallel load of scan chains, it is necessary to establish the instance names of all elements in the scan registers and to know whether there is inversion between each element and the scan data input or output. This is the information that **write_test** includes by default (*true*). If you want to exclude this information from the vector files, set the **write_test_include_scan_cell_info** variable to *false*.

To determine the current value of this variable, type **printvar write_test_include_scan_cell_info**. For a list of all **write_test** variables and their current values, type **print_variable_group write_test**.

SEE ALSO

write_test(2)
write_test_variables(3)

write_test_input_dont_care_value

Controls the logic value the **write_test** command outputs when you have an input with a don’t care condition.

TYPE

string

DEFAULT

X

GROUP

write_test_variables

DESCRIPTION

Controls the logic value the **write_test** command outputs when you have an input with a don’t care condition.

For example, in a design with multiple scan chains of unequal lengths, the serial input streams for the shorter chains need to be padded with arbitrary logic values.

This variable can have the value *1*, *0*, or *X* (also *x*). These values correspond to a don’t care condition of logic 1, logic 0, or logic don’t care, respectively. The default is *don’t care* (*X*).

For test equipment or test vector formats that do not support an input don’t care value, use this variable to assign logic 1 or logic 0.

To determine the current value of this variable, type **printvar write_test_input_dont_care_value**. For a list of all **write_test** variables and their current values, type **print_variable_group write_test**.

SEE ALSO

write_test(2)
write_test_variables(3)

write_test_max_cycles

Controls the automatic partitioning of long test sets across multiple files, by specifying the maximum number of tester cycles any one vector file can contain.

TYPE

integer

DEFAULT

0

GROUP

write_test_variables

DESCRIPTION

Controls the automatic partitioning of long test sets across multiple files, by specifying the maximum number of tester cycles any one vector file can contain. One tester cycle corresponds to one cycle of parallel (nonshift) operation of the device under test or to the shifting of scan data one bit in the scan chains.

Each vector file is self-contained; that is, the vectors within a specific file are independent of the state of the device under test and specifically do not require the vectors in other files to have been previously applied. This implies that **write_test_max_cycles** is used only for full-scan designs, and that test sets are not partitioned for partial scan designs. Additionally, the partitioning of the vector set has no effect on fault coverage.

By default, this variable is set to *0*, which implies there are no limits. Thus, automatic file partitioning is not performed. Work with your ASIC vendors to determine an appropriate value.

To determine the current value of this variable, type **printvar write_test_max_cycles**. For a list of all **write_test** variables and their current values, type **print_variable_group write_test**.

SEE ALSO

write_test_max_scan_patterns(3)
write_test_variables(3)
write_test_vector_file_naming_style(3)

write_test_max_scan_patterns

Controls the automatic partitioning of long test sets across multiple files, by specifying the maximum number of scan test patterns any one vector file can contain.

TYPE

integer

DEFAULT

0

GROUP

write_test_variables

DESCRIPTION

Controls the automatic partitioning of long test sets across multiple files, by specifying the maximum number of scan test patterns any one vector file can contain. The application of a scan test pattern encompasses the serial loading and unloading of the scan chains and consumes many tester cycles. Typically this correlates directly to the number of scan test patterns the **create_test_patterns** command generates. For designs with multiple system clocks, however, it might be necessary to repeat a number of times the application of each scan test pattern.

Each vector file is self-contained; that is, the vectors within a specific file are independent of the state of the device under test and do not require the vectors in other files to have been previously applied. This implies that **write_test_max_scan_patterns** is used only for full-scan designs, and that test sets are not partitioned for partial scan designs. Additionally, the partitioning of the vector set has no effect on fault coverage for full-scan designs.

By default, this variable is set to *0*, which implies there are no limits. Thus, automatic file partitioning is not performed. Work with your ASIC vendors to determine an appropriate value. Setting this variable to any value less than 2 results in no file partitioning.

To determine the current value of this variable, type **printvar write_test_max_scan_patterns**. For a list of all **write_test** variables and their current values, type **print_variable_group write_test**.

SEE ALSO

write_test_max_cycles(3)
write_test_variables(3)
write_test_vector_file_naming_style(3)

write_test_new_translation_engine

Specifies to the **write_test** command to choose to use new translation engine or old engine to do test program translation.

TYPE

string

DEFAULT

false

GROUP

write_test_variables

DESCRIPTION

Specifies to the **write_test** command to choose to use new translation engine or old engine to do test program translation.

When it is set to *true*, **write_test** will generate the test program in STIL format at first, then use new translation engine: Ltran and stil2wgl to translate the test program into other formats. Those formats supported by the new translation engine are: **stil**, **stil_testbench**, **wgl_serial**, **ftdl**, **tstl2**, **tdl91**.

When it is set to *false*, **write_test** will retrieve the test program from vdb file, translate it into sif file, then use Stran to translate it into other formats. Those formats supported by Stran are: **tdl91**, **synopsys**, **tds**, **wgl**, **verilog**, **vhdl**, **lsi**, **td**, **tstl2_scan**.

SEE ALSO

write_test(2)
write_test_variables(3)

write_test_pattern_set_naming_style

Specifies how to name pattern sets when long test sets are partitioned across multiple files.

TYPE

string

DEFAULT

TC_Syn_%d

GROUP

write_test_variables

DESCRIPTION

Specifies how to name pattern sets when long test sets are partitioned across multiple files. The pattern set name is a documentation aid and appears inside a comment header in the vector files.

The value of this variable is a format string containing one *%d* token that corresponds to the file number index. The file number index starts at 0 and, if the test set is partitioned, is incremented by 1 as additional files are needed.

To determine the current value of this variable, type **printvar write_test_pattern_set_naming_style**. For a list of all **write_test** variables and their current values, type **print_variable_group write_test**.

SEE ALSO

write_test_variables(3)

write_test_round_timing_values

Specifies to the **write_test** command to round to the nearest integer all timing values.

TYPE

string

DEFAULT

true

GROUP

write_test_variables

DESCRIPTION

Specifies to the **write_test** command to round to the nearest integer all timing values. The values are: input delay, output strobe time, bidir delay, clock period, and clock edge times. To achieve this result, set the value of this variable to *true* (the default).

If you do not want the timing values to be rounded to the nearest integer, set the value of this variable to *false*.

When this variable is set to *false*, the maximum resolution of the timing values generated by **write_test** is 0.01. For finer resolution, change the time units (for example, from ns to ps).

To determine the current value of this variable, type **printvar write_test_round_timing_values**. For a list of all **write_test** variables and their current values, type **print_variable_group write_test**.

SEE ALSO

write_test(2)
write_test_variables(3)

write_test_scan_check_file_naming_style

Specifies how to name the file containing the vectors that test the scan chain logic.

TYPE

string

DEFAULT

GROUP

write_test_variables

DESCRIPTION

Specifies how to name the file containing the vectors that test the scan chain logic.

The value of this variable is a format string containing two *%s* tokens. The first token pertains to the base name of the file specified by the *-output* option of the **write_test** command, or the value defaults to the design name. The second token is a file extension based on the targeted vector format, such as .v, for Verilog formatted vectors.

To determine the current value of this variable, type **printvar write_test_scan_check_file_naming_style**. For a list of all **write_test** variables and their current values, type **print_variable_group write_test**.

SEE ALSO

write_test(2)
write_test_max_cycles(3)
write_test_max_scan_patterns(3)
write_test_variables(3)

write_test_variables

SYNTAX

```
list write_test_formats = {"synopsys", "tssi_ascii", "tds", "verilog", "vhdl", "wgl
"}
string write_test_include_scan_cell_info = "true"
string write_test_input_dont_care_value = "X"
int write_test_max_cycles = 0
int write_test_max_scan_patterns = 0
string write_test_pattern_set_naming_style = "TC_Syn_%d"
string write_test_round_timing_values = "true"
string write_test_scan_check_file_naming_style = "%s_schk.%s"
string write_test_vector_file_naming_style = "%s_%d.%s"
string write_test_vhdlout = "textio"
string write_test_new_translation_engine = "false"
```

DESCRIPTION

These variables directly affect the **write_test** command.

To view this manual page online, type **man write_test_variables**. To view an individual variable description, type **man *var***, where *var* is the variable name.

write_test_formats
    Specifies the test vector formats recognized and created by the **write_test** command.

write_test_include_scan_cell_info
    Provides a mechanism to specify that the scan-chain/cell/inversion information should not be included in vector files. By default, this variable is set TRUE.

write_test_input_dont_care_value
    Controls the logic value output by the **write_test** command when you have an input with a "don't-care" condition.

write_test_max_cycles
    Allows the user to control the automatic partitioning of long test sets across multiple files by specifying the maximum number of tester cycles to be contained in any one vector file.

write_test_max_scan_patterns
    Allows the user to control the automatic partitioning of long test sets across multiple files by specifying the maximum number of scan-test patterns to be contained in any one vector file.

write_test_pattern_set_naming_style
    Specifies how pattern sets are named when long test sets are partitioned across multiple files.

write_test_round_timing_values
    When *true* (the default), **write_test** rounds all timing values (for example,

input delay, output strobe time, bidir delay, clock period and clock edge times) to the nearest integer. If you do not want the timing values to be rounded to the nearest integer, set the value of this variable to *false*.

write_test_scan_check_file_naming_style
: Specifies how to name the file containing the vectors which test the scan chain logic.

write_test_vector_file_naming_style
: Specifies how scan vector file names are derived, especially when long test sets must be split across multiple files.

write_test_vhdlout
: Determines whether or not **write_test -format vhdl** generates a VHDL test program in TEXTIO format. When this variable has the value of *textio*, the VHDL test program is written in TEXTIO format. When the variable has the value *inline* (the default), TEXTIO format is not used.

write_test_verilogout
: Determines whether or not **write_test -format verilog** or **write_test -format verilog -parallel** generates an LSI Logic- specific serial or parallel Verilog vector format. When this variable has the value *lsi*, the generated Verilog format will be specific to LSI Logic's test protocol. When the variable is unset (the default) or is set to any other value, the generated Verilog format will conform to the currently loaded test protocol.

write_test_wglout
: Determines whether or not **write_test -format wgl** generates LSI Logic-specific WGL vector format. When this variable has the value *lsi*, the generated WGL vector format will be specific to LSI Logic's test protocol. When the variable is unset (the default) or is set to any other value, the generated WGL format will conform to the currently loaded test protocol.

write_test_new_translation_engine
: Choose to use the new translation engine or the old one to do test program format translation. By default, this variable is set FALSE.

SEE ALSO

write_test(2)

write_test_vector_file_naming_style

Specifies how to name scan vector files, when long test sets are partitioned across multiple files.

TYPE

string

DEFAULT

%s_%d.%s

GROUP

write_test_variables

DESCRIPTION

Specifies how to name scan vector files, when long test sets are partitioned across multiple files.

The value of this variable is a format string containing *%s*, *%d*, and *%s* tokens, in this exact order. The first token, *%s*, pertains to the base name of the file specified by the *-output* option of the **write_test** command, or the value defaults to the design name. The second token, *%d*, corresponds to the file number index. The file number index starts at 0 and, if the test set is partitioned, is incremented by 1 as additional files are needed. The third *%s* token is a file extension based on the targeted vector format, such as .v for Verilog formatted vectors.

To determine the current value of this variable, type **printvar write_test_vector_file_naming_style**. For a list of all **write_test** variables and their current values, type **print_variable_group write_test**.

SEE ALSO

write_test(2)
write_test_max_cycles(3)
write_test_max_scan_patterns(3)

write_test_vhdlout

Determines whether the **write_test -format vhdl** command generates a VHDL test program in TEXTIO format.

TYPE

string

DEFAULT

inline

GROUP

write_test_variables

DESCRIPTION

The value you specify for the **write_test_vhdlout** variable determines whether the **write_test -format vhdl** command generates a VHDL test program in TEXTIO format.

If you set the value of this variable as **textio**, the tool writes the VHDL test program in TEXTIO format. If you set the value to **inline** (the default), TEXTIO format is not used.

Compiling a large number of patterns (and placing them in memory) can sometimes cause swapping during simulation. Using TEXTIO format is helpful in such a situation. Both serial load and parallel load testbench generation are supported. To generate a parallel loadable testbench, use the **write_test -parallel** command.

To see the current value of this variable, type

```
psyn_shell-xg-t> printvar write_test_vhdlout
```

For a list of all **write_test** variables and their current values, see the **write_test_variables**(3) man page.

SEE ALSO

write_test(2)
write_test_variables(3)

x11_set_cursor_background

Specifies background color of the cursor in the Design Analyzer menus and viewer.

TYPE

string

DEFAULT

""

GROUP

view_variables

DESCRIPTION

Specifies background color of the cursor in the Design Analyzer menus and viewer. This variable can be set in the **.synopsys_dc.setup** initialization file. Any color in **/usr/lib/X11/rgb.txt** is valid.

To determine the current value of this variable, type **printvar x11_set_cursor_background**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_variables(3)

x11_set_cursor_foreground

Specifies foreground color of the cursor in the Design Analyzer menus and viewer.

TYPE

string

DEFAULT

magenta

GROUP

view_variables

DESCRIPTION

Specifies foreground color of the cursor in the Design Analyzer menus and viewer. This variable can be set in the **.synopsys_dc.setup** initialization file. Any color in **/usr/lib/X11/rgb.txt** is valid.

To determine the current value of this variable, type **printvar x11_set_cursor_foreground**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_variables(3)

x11_set_cursor_number

Specifies the cursor, from the standard X cursor font used by the Design Analyzer menus and viewer.

TYPE

integer

DEFAULT

-1

GROUP

view_variables

DESCRIPTION

Specifies the cursor, from the standard X cursor font used by the Design Analyzer menus and viewer.

If this variable is not set, or is set to *-1* (the default), the cursor in the X background is also used for all windows and menus. Set this variable in the **.synopsys_dc.setup** initialization file.

To determine the current value of this variable, type **printvar x11_set_cursor_number**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_variables(3)

xterm_executable

Specifies the path to an xterm program spawned to run Synopsys analysis tools (for example, RTL Analyzer or BCView). The default is *xterm*.

TYPE

string

DEFAULT

xterm

GROUP

bc_variables
hdl_variables

DESCRIPTION

Specifies the path to an xterm program spawned to run Synopsys analysis tools (for example, RTL Analyzer or BCView). The default is *xterm*.

If an xterm is not found in your path, set this variable to point to an xterm executable. For example, if your xterm is not in your $path, but is located at /usr/bin/X11/xterm, set this variable as follows:

```
dc_shell> xterm_executable = /usr/bin/X11/xterm
```

To determine the current value of this variable, type **printvar xterm_executable**. For a list of all **bc** or **hdl** variables and their current values, type **print_variable_group bc** or **print_variable_group hdl**.

SEE ALSO

bc_view (2), **rtl_analyzer** (2); **bc_variables** (3), **hdl_variables** (3).